ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Sistema móvil para la gestión de vehículos

David Borrego Gutiérrez
Manuel Palomo Duarte
Lorena Gutiérrez Madroñal

Índice general

Índice de figuras

Índice de cuadros

A mi familia y en especial a mis padres.

Capítulo 1

Introducción

La realización de este proyecto tiene como objetivo la aplicación de conocimientos adquiridos durante el estudio de la titulación de Ingeniería Técnica en Informática de Gestión, el descubrimiento y aplicación de nuevas tecnologías para profundizar en el conocimiento del desarrollo del software.

1.1. Motivación

Mirando a nuestro alrededor, se puede ver cómo todos nuestros útiles del día a día se encuentran en continuo cambio, uno de los mercados con más cambios en la actualidad es el ámbito de la automoción. En este momento la automoción está viviendo un gran giro, nos encontramos ante nuevos vehículos totalmente automatizados, con cantidad de información, conectados a internet, etc.

En la pelea por descatar en este mundo, vemos nuevas aplicaciones con las que los fabricantes intentan destacar en el mercado. Facilitar predicciones, datos de consumo, etc. Esta es una forma fácil de ofrecer funcionalidades pero... ¿Qué pasa con esas personas que tiene un coche antiguo?.

Si se realiza una búsqueda de aplicaciones que nos permitan llevar un control sobre vehículos, rápidamente se ve que no existe ninguna aplicación sencilla y centralizada dedicada a este fin. De aquí surge la idea de englobar varias tecnologías para crear un sistema, el «Sistema móvil para la gestión de vehículos».

Gracias a la aplicación podemos encontrar un nuevo punto de unión con gran cantidad de control de datos e infinidad de nuevas posibilidades a futuro ya que todos los datos se encuentran en una misma base de datos.

1.2. Sistemas actuales

En la actualidad, existen sistemas parecidos pero son propiedad de las marcas, es decir, cuando se compra un vehículo hay marcas que nos proporcionan algunos datos estadísticos. Los datos son pocos ya que no es fácil conectar un vehículo a Internet y con una conexión estable.

Otro problema de este tipo de sistemas es que no evoluciona, son sistemas instalados en el vehículo y cuya versión permanecerá estable durante toda la vida del mismo.

Del mismo modo, existe la posibilidad de personas que tengan más de un vehículo de distinta marca, siendo imposible la comparación entre vehículos o la centralización de los datos.

Si se realiza una visualización del mercado más global, las compañías no comparten este tipo de información ya que no les conviene que exista una base de datos con datos de distintas marcas ya que se podrían producir comparaciones de consumo, precios, etc.

1.3. Objetivos y alcance del proyecto

El objetivo principal del presente proyecto es la centralización de datos. Se presenta un proyecto ambicioso a largo plazo con infinidad de posibilidades. En la actualidad el primer objetivo es proporcionar un punto de entrada de datos desde una aplicación móvil para Android.

Para poder satisfacer las necesidades de este primer y principal objetivo se han cumplido los siguientes objetivos:

- El sistema proporciona una interfaz para los usuarios:
  - Proporciona toda una interfaz de registro, modificación de usuarios.
  - Todos los servicios de la aplicación se encuentran securizados, por tanto la información es segura y privada.
  - El almacenamiento no se realiza en ninguna aplicación externa como la aplicación Android, sino que se realizada en el servidor. De esta forma la información es más segura y no generarán perdidas.
- Gestión de todas las rutas o repostajes realizados por un determinado vehículo y usuario. En caso de desearlo, el usuario podrá hacer uso del sistema de gestión de rutas. De esta forma se irán actualizando los kilómetros del vehículo y quedará registrado un historial de trayectos. Estos trayectos guardarán información sobre la fecha en que se realizó, combustible utilizado, su precio, consumo medio, etc.
- Gestión de varios vehículos e historial de vida del mismo.
  - Los usuario pueden gestionar de forma independiente cualquiera de sus vehículos.
- Gestión de todas las piezas del vehículo, así como su historial de mantenimiento, modificaciones, compras, etc.
  - El sistema provee un apartado de gestión de piezas, desde aquí se podrán ir registrando las piezas, historial de mantenimientos de las mismas, etc.
  - Ádemás, el sistema avisará y realizará cálculos sobre la fecha estimada de cambios de las piezas, su precio, etc.

Todas estas posibilidades se ofrecen desde la aplicación móvil pero son almacenadas a través de otra aplicación. Esta aplicación realiza las funciones de servidor, y actúa como punto de entrada, es decir, es aquí donde se encuentran almacenados los datos.

Gracias a tener todos los datos almacenados y centralizados en un servidor REST API el alcance del proyecto a largo plazo se muentra muy ambicioso. Por ejemplo, sería posible realizar estadísticas de forma anónima realizando comparaciones entre marcas.

1.4. Organización

En el presente documento se facilita una introducción al PFC. En esta introducción se habla del objetivo, alcance, etc.

Posteriormente se relata el análisis, diseño, codificación y demás aspectos relacionados con la creación de la aplicación.

Al final, se detallan los manuales y documentación de ambas aplicaciones.

Adjunto, podemos encontrar las dos aplicaciones:

- **.APK** Aplicación para Android.
- **.WAR** Aplicación REST API.

1.5. Acrónimos y definiciones

En este apartado se encuentran las definiciones de los acrónimos utilizados durante todo el PFC, frameworks o palabras técnicas y específicas.

PFC: Proyecto Final de Carrera.

APK (Application Package File): Es la extensión de los JAR de las aplicaciones para Android en lenguaje nativo. Con este tipo de ficheros podemos realizar la instalación de una aplicación en un dispositivo Android.

WAR (Web Application Archive): Es un tipo de JAR utilizado para empaquetar un conjuto de JSP, Servlets, XML, HTML, JS o cualquier otro tipo necesario para una aplicación Web.

JAR: Formato de archivo usado para empaquetar todos los componentes necesarios en una aplicación Java.

JSP(Java Server Pages): Tecnología orientada a crear páginas web con programación en Java.

Servlet: Clase Java utilizada para ampliar las capacidades de un servidor. Son comunmente conocidos por estar alojados en servidores web pudiendo acceder a ellos a través de llamadas web.

REST API: Tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP. Es un estándar sencillo que trabaja con JSON normalmente. Define un uso estructura de URIs usando correctamente los verbos HTTP.

Spring: Amplio Framework de código abierto para Java. Spring se compone de muchos módulos que facilitan el desarrollo de todo tipo de aplicaciones Java.

Spring Security: Módulo de Spring dedicado a la securización de aplicaciones, principalmente a aplicaciones web. Definiendo un sencillo xml de configuración y a través de un sistema de roles podemos decidir qué usuarios tienen acceso a qué partes de la aplicación.

Spring Core: Módulo de Spring dedicado a la inversión de control e inyección de dependencias. Este módulo es el más conocido de Spring.

Spring MVC: Módulo orientado al manejo de la vista, tanto con JSP, JSF o cualquier otro tipo de tecnología. También es muy conocido por la facilidad a la hora de realizar servicios REST.

Android SDK: Conjuto de herramientas de desarrollo que nos permiten crear aplicaciones para Android.

AWS (Amazon Web Services): Conjunto de servicios en la nube que proporciona Amazon, pagando únicamente lo que utilicemos. Este tipo de infraestructuras nos aseguran una alta disponibilidad con capacidad de escalabilidad sin necesidad de tener nuestros propios servidores.

MySQL: Base de datos relacional y gratuita. Esta base de datos ha sido la utilizada durante todo el proyecto.

ORM (Object-Relational mapping): Un ORM es normalmente una técnica o framework que se encarga de convertir en clases y viceversa la información alojada en una base de datos.

POST: Verbo para realizar las llamadas HTTP referentes a la creación de contenidos.

PUT: Verbo HTTP utilizado para la actualización de objetos.

GET: Verbo HTTP utilizado para la recuperación de datos.

DELETE: Verbo utilizada en las llamadas HTTP que se encargan de eliminar contenidos.

APP Engine: Similar a la definición anterior de AWS pero de Google.

DAO(Data Access Objects): Es una interfaz orientada a objetos. En J2EE se utiliza para los accesos a base de datos.

JSESSONID: Cookie necesaria para comprobar el login de usuario. El sistema es sencillo, cuando un usuario se loga en el sistema, este le devuelve una cookie que será necesaria para cada una de las siguientes llamadas.

SQLite: Base de datos relacional, pequeña y embebida en aplicaciones Android.

MVC: Patrón de diseño comunmente conocido como modelo-vista-controlador. En este patrón de diseño destaca una capa con el modelo, una capa que controla la vista e interactua con el modelo y una capa que se encarga de la vista.

Factory: Patrón de diseño que consiste en utilizar una clase abstracta con unos cuantos métodos definidos y otros sin definir. Cuando esta clase es heredada por una determinada entidad adquiere un comportamiento específico, teniendo implementado gran cantidad del código.

Singleton: Patrón de diseño que destaca por tener únicamente una instancia de cada servicio.

Controller: Un controller es una clase en Spring MVC que se encarga de definir un conjunto de Servicios REST.

Capítulo 2

Planificación

En este capítulo son detallados todos los aspectos relativos a la planificación del proyecto.

En primer lugar, se puede ver la metodología utilizada durante todo el proceso de desarrollo de la aplicación y su descomposición en fases.

Posteriormente, es definida ampliamente la planificación del proyecto. Dentro de esta planificación se encuentran descritas de forma detallada las estimaciones de tiempo para cada una de las tareas a realizar separadas en subsistemas. Todas las tareas están planificadas para un perfil en concreto. Con todos estos datos se ha generado un diagrama de Gantt con la planificación completa.

Para finalizar con el capítulo, se encuentra una estimación de coste conforme a la estimación de tiempo indicada.

2.1. Metodología de desarrollo

Para el desarrollo de la aplicación ha sido empleado un ciclo de desarrollo en espiral. Este modelo es el que más se ajusta a las necesidades de este desarrollo ya que es la mejor forma de ir evolucionando la aplicación y añadiendo nuevas funcionalidades.

El desarrollo se ha dividido en varias iteraciones que han ido completando las funcionalidades tanto en la parte del servidor como en la del cliente. Todas las iteraciones han tenido una duración aproximada de 2 meses. Debido a no disponer de tiempo completo para el desarrollo se ha considerado en la planificación un día como 2 horas de desarrollo, una semana como 14 horas y un mes como 30 días.

Las fases o iteraciones de forma global han sido las siguientes:

- **Fase de análisis:** Se realiza el estudio de la aplicación, las necesidades y funcionalidades necesarias. Una vez recogidos todos los requisitos se procede al estudio de las tecnologías disponibles para cubrir las necesidades.
- **Creación estructura base:** Se crea la arquitectura de la aplicación. Se realiza la configuración de una versión inicial del servidor y cliente, si configuran todos los Framework a utilizar, etc.
- **Fase de usuarios:** Se realizan las operaciones y funcionalidades relacionadas con los usuarios.
- **Fase de vehículos:** Se añaden todas las funcionalidades relacionadas con los vehículos.
- **Fase rutas:** Todas las operaciones relacionadas con las rutas de la aplicación.
- **Fase piezas:** Se añaden las funcionalidades relacionadas con las piezas.

A parte, fuera del desarrollo del producto, se ha realizado de forma paralera las presentes memorias del PFC.

2.2. Organización

El proyecto ha sido realizado únicamente por una persona que ha desempeñado unos cuatro papeles diferentes.

Es importante destacar que toda la planificación hubiese sido diferente y el resultado hubiése sido obtenido mucho antes ya que muchas de las tareas podrían haber sido paralelizadas. Debido a que sólo ha habido una persona, las tareas se han ido entrelazando, no pudiendo comenzar una hasta que no terminaba otra.

Los papeles o roles desempeñados han sido los siguientes:

- **Analista:** Encargado de todo el análisis de la aplicación, decisión de tecnologías, etc. También responsable de la creación de la arquitectura de la aplicación.
- **Programador Java:** Encargado del desarrollo de toda la parte del Servidor.
- **Programador Android:** Encargado del desarrollo de la aplicación Cliente.
- **Técnico de Sistemas:** Encargado de la infraestructura de la aplicación. Decisión de la infraestructura más adecuada, configuración, etc.

2.3. Planificación del proyecto

Como se ha indicado en el apartado anterior, esta planificación es para una persona que realiza cuatro roles diferentes. En caso de haber trabajado en el proyecto cuatro personas, la estimación sería mucho más reducida.

2.3.1. Estimación de tiempo

Para el desarrollo se han modificado las jornadas «normales» de trabajo. Se ha considerado una semana productiva como una semana de 7 días en los que se trabaja de media una hora diaria de lunes a viernes y cinco horas sábados y domingos. De esta forma se obtiene una jornada de 15 horas semanales de media.

Para la estimación de tareas se ha tenido en cuenta un día como 2 horas, ya que es la media apróximada de tiempo empleado diariamente, incluyendo fines de semana.

Con estos tiempos la estimación obtenida es la siguiente:

La planificación del Servidor y el Cliente se alargan ya que se producen retrasos en los proyectos debido a que partes del cliente no pueden terminar hasta que no terminen en el servidor y viceversa.

Nombre	Duración	Inicio	Terminado	Predecesores
⊟Servidor	466 days	**1/10/13 8:00**	**10/12/14 21:00**	
Análisis	20 days	1/10/13 8:00	20/10/13 9:00	
⊟Creación estructura base	41 days	**20/10/13 9:00**	**27/11/13 21:00**	
Crear proyectos Maven	7 days	20/10/13 9:00	26/10/13 13:00	2
Configurar Spring Core	5 days	27/10/13 8:00	1/11/13 21:00	4
Configurar Spring MVC	10 days	2/11/13 8:00	9/11/13 13:00	5
Configurar Spring Security	10 days	10/11/13 8:00	17/11/13 13:00	6
Configurar Hibernate	7 days	18/11/13 20:00	24/11/13 12:00	7
Subir a GIT	2 days	24/11/13 12:00	27/11/13 21:00	8
⊟Usuarios	15 days	**1/02/14 11:00**	**15/02/14 11:00**	
Modelo, DAO, Servicios	10 days	1/02/14 11:00	9/02/14 11:00	29
Servicios REST, DTO	5 days	9/02/14 11:00	15/02/14 11:00	11
⊟Vehículos	14 days	**22/03/14 10:00**	**4/04/14 21:00**	
Modelo, DAO, Servicios	10 days	22/03/14 10:00	30/03/14 10:00	33
Servicios REST, DTO	4 days	30/03/14 10:00	4/04/14 21:00	14
⊟Rutas	21 days	**11/05/14 10:00**	**31/05/14 12:00**	
Modelo, DAO, Servicios	14 days	11/05/14 10:00	24/05/14 13:00	38
Servicios REST, DTO	7 days	25/05/14 8:00	31/05/14 12:00	17
⊟Piezas	25 days	**12/07/14 8:00**	**2/08/14 13:00**	
Modelo, DAO, Servicios	15 days	12/07/14 8:00	25/07/14 21:00	43
Servicios REST, DTO	10 days	26/07/14 8:00	2/08/14 13:00	20
Corrección Bugs	25 days	16/11/14 11:00	10/12/14 21:00	54

Figura 2.1: Planificación del Servidor REST API

Nombre	Duración	Inicio	Terminado	Predecesores
Cliente Android	435 days	**28/11/13 20:00**	**7/01/15 21:00**	
Crear estructura base	70 days	**28/11/13 20:00**	**1/02/14 11:00**	
Creación pantalla principal	10 days	28/11/13 20:00	7/12/13 11:00	9
Creación ORM	15 days	7/12/13 11:00	21/12/13 11:00	25
Creación API para peticiones	10 days	21/12/13 11:00	29/12/13 11:00	26
Creación HTTP Client propio	30 days	29/12/13 11:00	26/01/14 11:00	27
Creación menú y enlaces	5 days	26/01/14 11:00	1/02/14 11:00	28
Usuarios	37 days	**15/02/14 11:00**	**22/03/14 10:00**	
Registro usuarios	15 days	15/02/14 11:00	1/03/14 11:00	12
Login usuarios	15 days	1/03/14 11:00	15/03/14 11:00	31
Servicios en API para Usuarios	7 days	15/03/14 11:00	22/03/14 10:00	32
Vehículos	41 days	**5/04/14 8:00**	**11/05/14 10:00**	
Pantalla listado	15 days	5/04/14 8:00	18/04/14 21:00	15
Pantalla individual	15 days	19/04/14 8:00	2/05/14 21:00	35
Servicios en API	7 days	3/05/14 8:00	8/05/14 21:00	36
Añadir Spinner pantalla principal	4 days	9/05/14 20:00	11/05/14 10:00	37
Rutas	43 days	**31/05/14 12:00**	**11/07/14 21:00**	
Pantalla listado	15 days	31/05/14 12:00	14/06/14 12:00	18
Pantalla individual	18 days	14/06/14 12:00	29/06/14 13:00	40
Paso de parámetros entre pantallas	3 days	30/06/14 20:00	5/07/14 9:00	41
Servicios en API	7 days	5/07/14 9:00	11/07/14 21:00	42
Piezas	79 days	**3/08/14 8:00**	**15/10/14 21:00**	
Pantallas	15 days	3/08/14 8:00	16/08/14 13:00	21
Pantalla individual	15 days	17/08/14 8:00	30/08/14 13:00	45
Estimaciones	10 days	31/08/14 8:00	7/09/14 13:00	46
Cambios	21 days	8/09/14 20:00	28/09/14 10:00	47
Servicios en API	15 days	28/09/14 10:00	12/10/14 10:00	48
Paso de parámetros entre pantallas	3 days	12/10/14 10:00	15/10/14 21:00	49
Corrección de Bugs	30 days	11/12/14 20:00	7/01/15 21:00	22

Figura 2.2: Planificación del Cliente Android

Nombre	Duración	Inicio	Terminado	Predecesores
Implantación del Sistema	35 days	**16/10/14 20:00**	**16/11/14 11:00**	
Configuración Balanceador	20 days	16/10/14 20:00	2/11/14 11:00	50
Configuración Nodos	15 days	2/11/14 11:00	16/11/14 11:00	53

Figura 2.3: Planificación de la configuración de la infraestructura necesaria.

2.3.2. Diagramas de Gantt

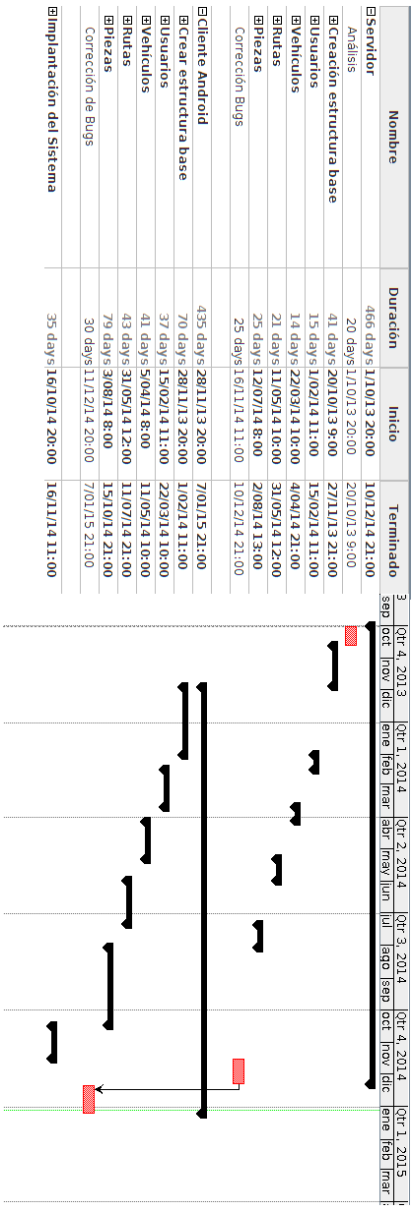

Figura 2.4: Diagrama de Gantt

2.4. Costes

Para la estimación de costes he utilizado las Tablas salariales 2010 IV Convenio Colectivo de CCOO [CCOO(2010)] .

Los perfiles necesarios son cuatro, un Analista, un Programador Java, un Programador Android y un Técnico de Sistemas que realice la instalación del sistema.

Tal y como podemos apreciar en el salario para un analista en las tablas, vemos un puesto de grupo I, T. S. Informática. A continuación calculo el coste por hora de un trabajador de características similares quién desempeñará las funciones de Analista:

$$\frac{(24{,}092{,}25 + 11{,}548{,}56)}{(12m \cdot 22d \cdot 8h)} = 16{,}87\frac{e}{h}$$

Tanto el Programador J2EE como el Programador Android son puestos que se asemejan en las tablas a Técnico Especialista, del grupo III, por tanto su salario a la hora sería:

$$\frac{(17{,}756{,}40 + 5{,}733{,}84)}{(12m \cdot 22d \cdot 8h)} = 11{,}80\frac{e}{h}$$

El especialista en instalación y configuración de sistemas lo encajo en el grupo II en un T. G. M. Informática:

$$\frac{(20{,}298{,}30 + 9{,}199{,}56)}{(12m \cdot 22d \cdot 8h)} = 13{,}96\frac{e}{h}$$

Una vez tenemos calculado el coste por hora de cada uno de los perfiles, sólo quedaría multiplicar este coste por la estimación de horas realizada en cada una de las tareas y reflejadas en el diagrama de Gantt del apartado anterior. Podemos ver el resumen de todos los cálculos mencionados en la siguiente tabla:

Tarea	Analista	P.J2EE	P.Android	Téc. Sistemas	Total
S - Análisis	674,80 €	0 €	0 €	0 €	674,8 €
S - Arquitectura base	1.383,34 €	0 €	0 €	0 €	1.383,34 €
S - Usuarios	0 €	354 €	0 €	0 €	354 €
S - Vehículos	0 €	330,4 €	0 €	0 €	330,4 €
S - Rutas	0 €	495,60 €	0 €	0 €	495,60 €
S - Piezas	0 €	590 €	0 €	0 €	590 €
S - Corrección bugs	0 €	590 €	0 €	0 €	590 €
C - Arquitectura base	2.361,80 €	0 €	0 €	0 €	2.361,80 €
C - Usuarios	0 €	0 €	873,20 €	0 €	873,20 €
C - Vehículos	0 €	0 €	967,60 €	0 €	967,60 €
C - Rutas	0 €	0 €	1.014,80 €	0 €	1.014,80 €
C - Piezas	0 €	0 €	1.864,40 €	0 €	1.864,40 €
C - Corrección bugs	0 €	0 €	708 €	0 €	708 €
Implantanción	0 €	0 €	0 €	977,2 €	977,2 €
Total	4.419,94 €	2.360 €	5.428 €	977.2 €	**13.185,14 €**

Como conclusión tenemos un coste total de 13.185,14 € para la estimación realizada.

Capítulo 3

Análisis de Requisitos

3.1. Catálogo de actores

Actualmente sólo existe un actor en la aplicación. Todas las operaciones a realizar desde el cliente únicamente se pueden realizar como usuario registrado.

3.2. Requisitos funcionales

3.2.1. Casos de usos

3.2.1.1. Esquema general

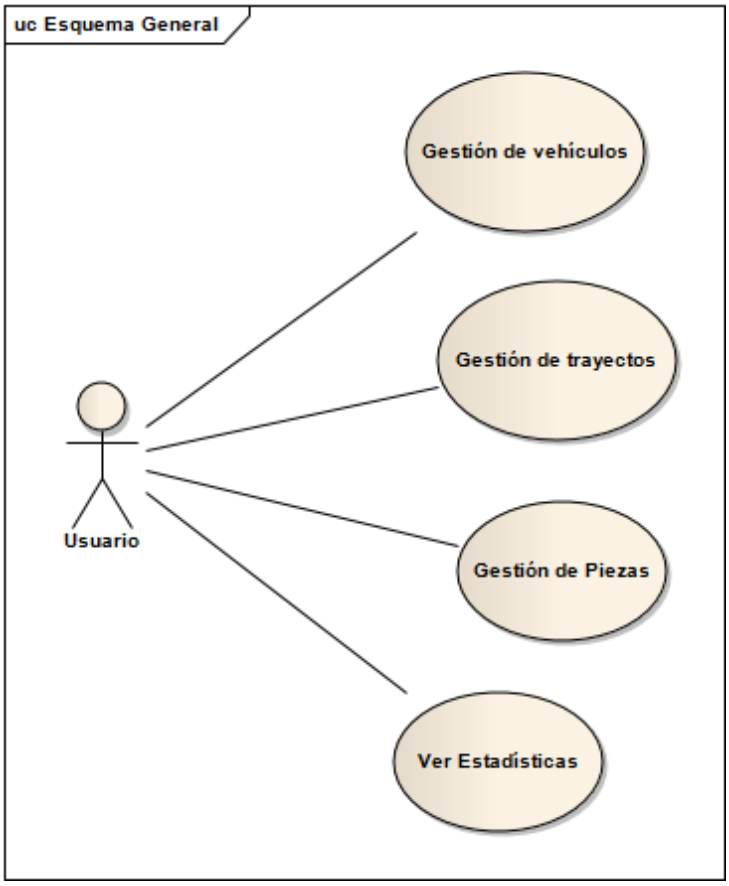

Figura 3.1: Diagrama del Caso de Uso general de la aplicación

3.2.1.2. CU Gestión Usuarios

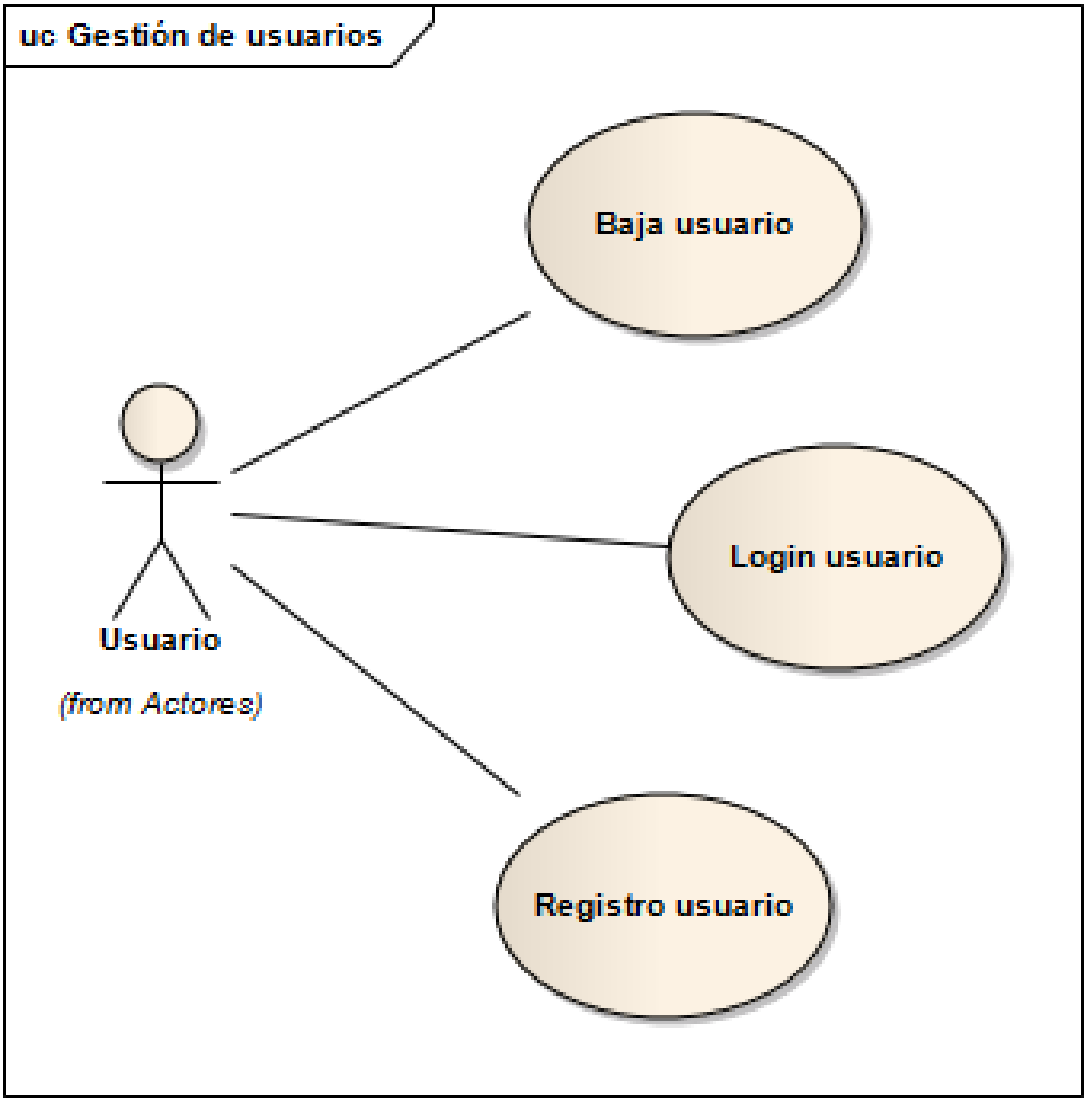

Figura 3.2: Diagrama del Caso de Uso Gestión Usuarios

3.2.1.3. CU Gestión de vehículos

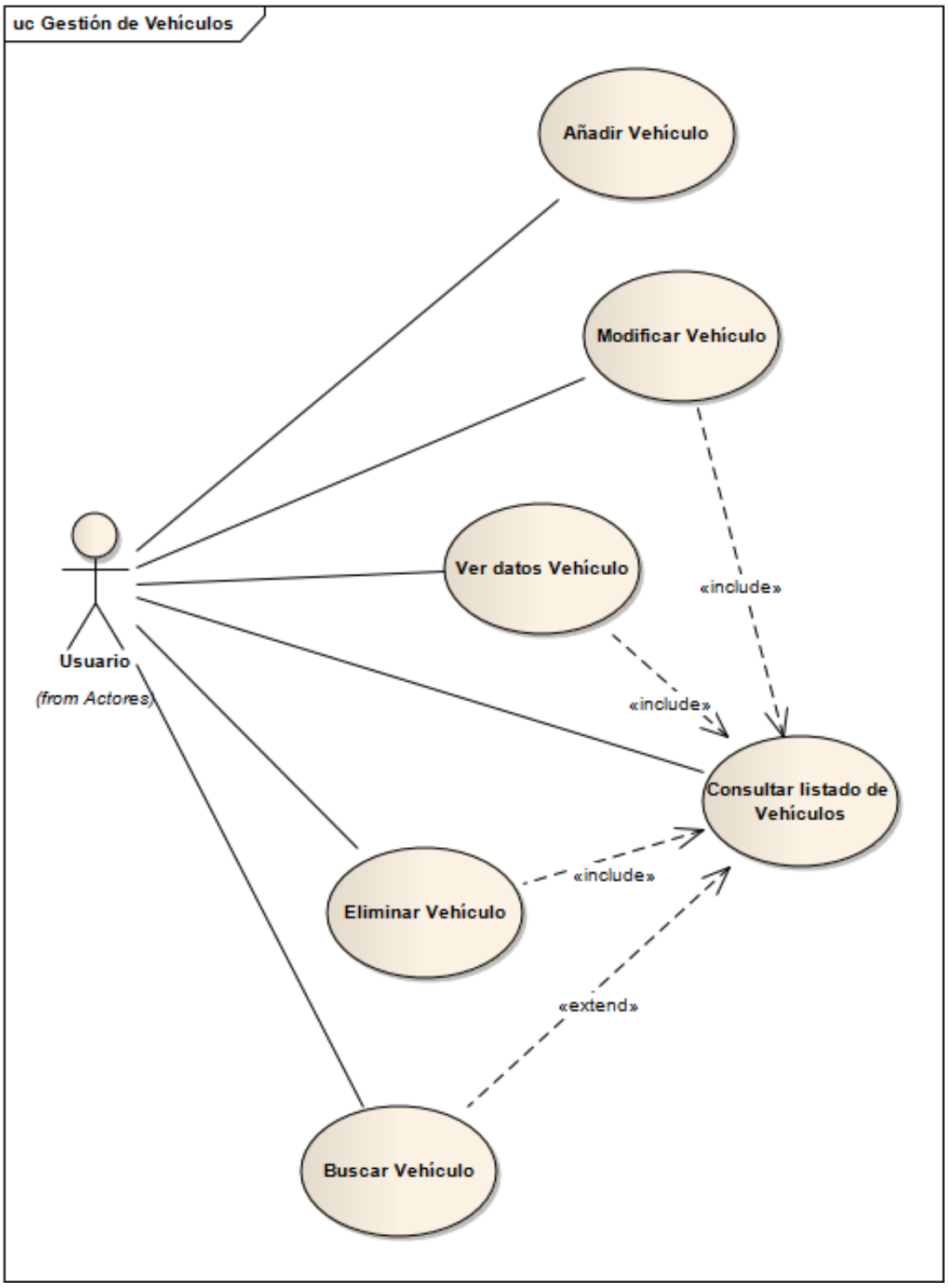

Figura 3.3: Diagrama del Caso de Uso Gestión de vehículos

3.2.1.4. CU Gestión de trayectos

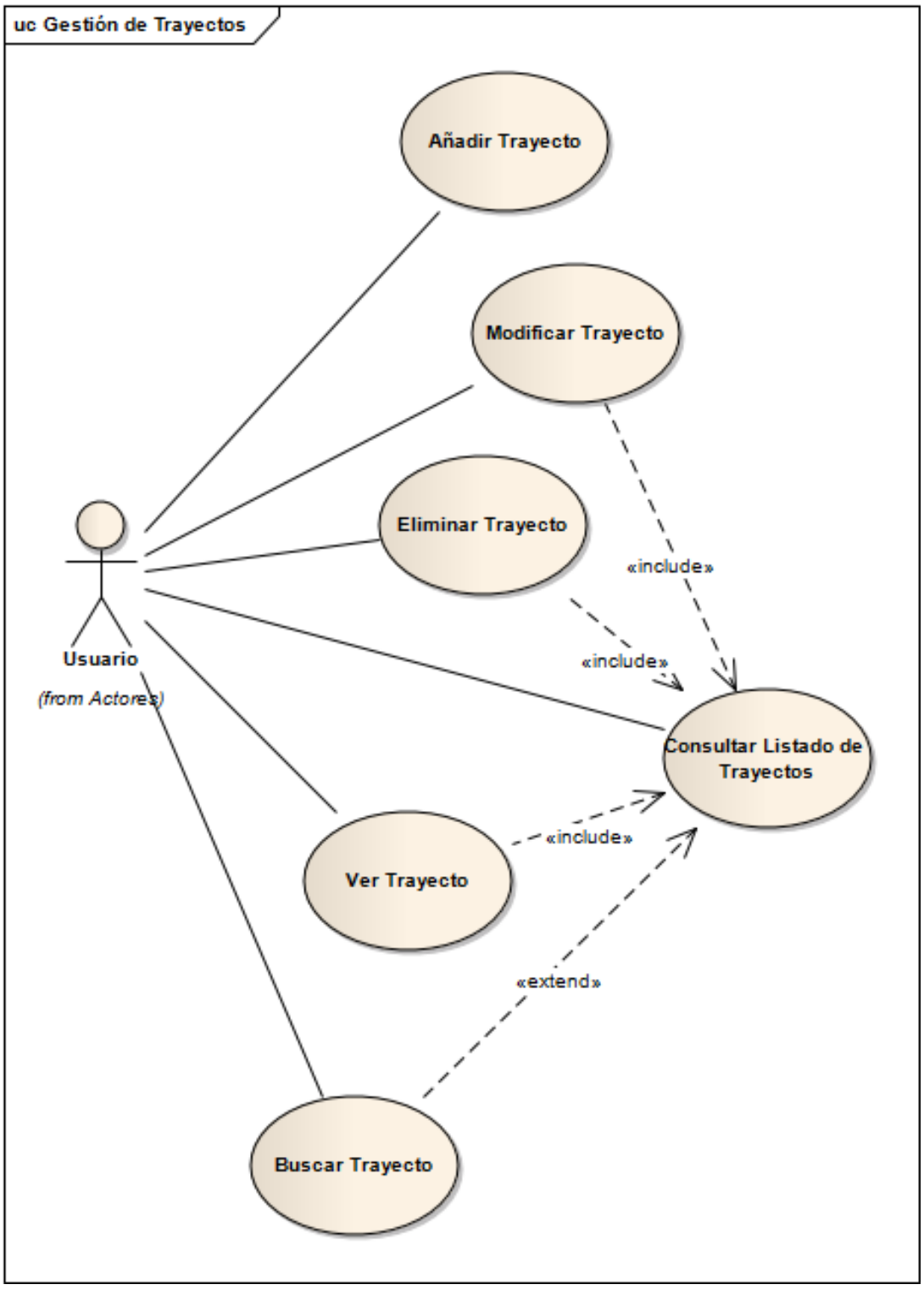

Figura 3.4: Diagrama del Caso de Uso Gestión de trayectos

3.2.1.5. CU Gestión de piezas

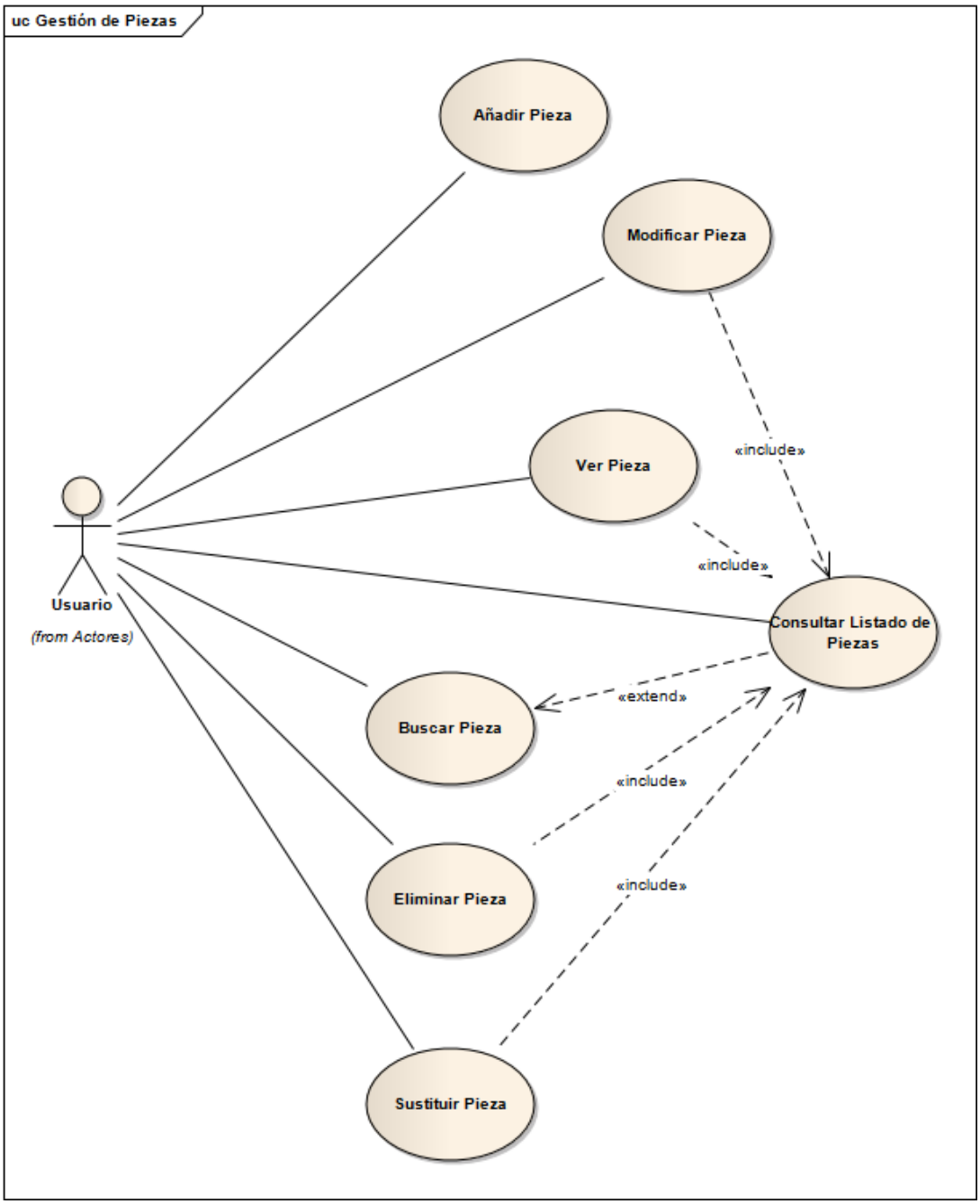

Figura 3.5: Diagrama del Caso de Uso Gestión de piezas

3.2.1.6. CU Consultar estadísticas

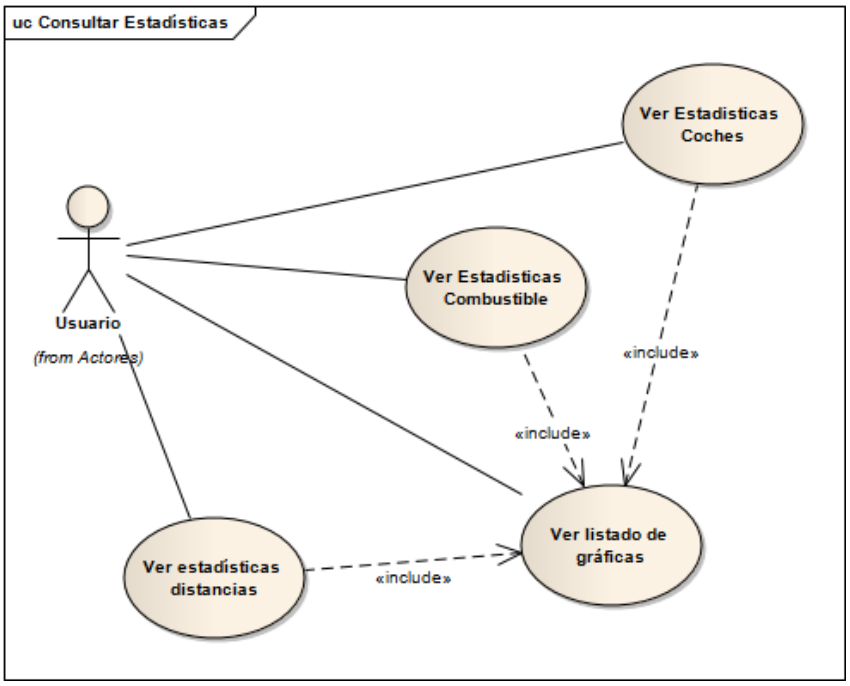

Figura 3.6: Diagrama del Caso de Uso Consultar estadísticas

3.2.2. Especificación de los casos de uso

3.2.2.1. CU Gestión Usuarios

3.2.2.1.1 CU Baja Usuario

Caso de uso:	**Baja Usuario**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema.
Postcondición:	- El usuario será eliminado del sistema.

Curso normal

1. El usuario desea dar de baja su cuenta en el sistema.
2. El usuario sobre «Dar de baja».
3. El sistema pide confirmación.
4. El sistema valida que el usuario se encuentra logueado en el sistema.
5. El sistema realiza la baja del usuario y todos sus datos relacionados.
6. El usuario es redirigido a la pantalla principal del sistema.

Curso alternativo

1-3a. El usuario cancela el proceso.

1. El sistema cancela y redirecciona a la pantalla principal.

2a. El usuario no confirma.

1. El muestra al usuario la pantalla principal.

4a. El usuario no se encuentra logueado.

1. El sistema muestra el mensaje de error «Necesita estar logueado»

3.2.2.1.2 CU Login Usuario

Caso de uso:	**Login usuario**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado en el sistema.
Postcondición:	- El usuario es logueado en el sistema.

Curso normal

1. El usuario desea loguearse en el sistema.
2. El sistema recupera su nombre de usuario y contraseña de BBDD y realiza el logueado del usuario.
3. El usuario es logueado en el sistema.
4. El usuario ve la pantalla principal con sus datos actualizados.

Curso alternativo

2a. Los datos del usuario no se encuentra en la BBDD local de la aplicación.

1. El sistema muestra la pantalla para introducir nombre de usuario y contraseña.

3a. Las credenciales no son correctas.

1. El sistema muestra un listado vacío con el mensaje «La credenciales introducidas no son correctas»
2. El sistema muestra la pantalla para introducir nombre de usuario y contraseña.

3.2.2.1.3 CU Registro usuario

Caso de uso:	**Registro usuario**
Actores:	- Usuario
Precondición:	- El usuario no se encuentra registrado ni logueado en el sistema.
Postcondición:	- El usuario quedará registrado en el sistema.

Curso normal

1. El usuario darse de alta en el sistema.
2. El sistema muestra la pantalla de registro de usuario.
3. El usuario introduce los datos solicitados.
4. El usuario pulsa sobre el botón «registrar».
5. El sistema valida los datos.
6. El sistema registra el usuario.
7. El sistema redirige al usuario a la pantalla principal.

Curso alternativo

1-4a. El usuario cancela el proceso.

1. El sistema no realiza ningún cambio.
2. El usuario es redirigido al caso de uso «Pantalla principal»

5a. Falta algún dato.

1. El sistema muestra el mensaje de error «Algún campo se encuentra vacío»
2. El sistema marca en rojo el campo vacío.

5b. Algún dato es incorrecto

1. El sistema muestra el mensaje de error «Error al validar los datos»
2. El sistema marca en rojo el campo.

3.2.2.2. CU Gestión de vehículos

3.2.2.2.1 CU Añadir Vehículo

Caso de uso:	**Añadir vehículo**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema.
Postcondición:	- Un nuevo vehículo será creado y relacionado con el usuario.

Curso normal

1. El usuario desea crear un nuevo vehículo.
2. El usuario presiona sobre crear nuevo producto.
3. El sistema valida que el usuario se encuentra logueado en el sistema y no ha sobrepasado el límite de vehículos disponible.
4. El usuario introduce los datos del nuevo vehículo y presiona en guardar.
5. El sistema valida los datos introducidos.
6. El sistema guarda los datos del nuevo vehículo y los relaciona al usuario.
7. El usuario es redirigido a la pantalla «Gestión de Vehículos».

Curso alternativo

1-6a. El usuario cancela el proceso.

1. El sistema cancela y redirecciona al listado de vehículos.

3a. El usuario no se encuentra logueado.

1. El sistema muestra el mensaje de error «Necesita estar logueado»
2. El sistema cancela y redirecciona al listado de vehículos.

3b. El usuario ha sobrepasado el límite de vehículos.

1. El sistema muestra el mensaje de error «Ha sobrepasado el límite de vehículos»

5a. Falta algún dato.

1. El sistema muestra el mensaje de error «Algún campo se encuentra vacío»
2. El sistema marca en rojo el campo vacío.

5b. Algún dato es incorrecto

1. El sistema muestra el mensaje de error «Error al validar los datos»
2. El sistema marca en rojo el campo.

3.2.2.2.2 CU Buscar Vehículo

Caso de uso:	**Buscar vehículo**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema.
Postcondición:	- El usuario obtiene el resultado de su búsqueda.

Curso normal

1. El usuario desea buscar un vehículo.
2. El sistema muestra un listado con sus vehículos.
3. El usuario filtra por los campos requeridos.
4. El usuario obtiene el resultado de su búsqueda.
5. El usuario selecciona el vehículo buscado.

Curso alternativo

1-5a. El usuario cancela el proceso.

1. El sistema redirige al usuario a la pantalla principal.

5a. La búsqueda no devuelve ningún resultado.

1. El sistema muestra un listado vacío con el mensaje «No se han encontrado vehículos con el filtro seleccionado»

3.2.2.2.3 CU Modificar Vehículo

Caso de uso:	**Modificar vehículo**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El vehículo a modificar existe. - El usuario es el dueño del vehículo a modificar.
Postcondición:	- Los datos del vehículo son modificados.

Curso normal

1. El usuario desea modificar un vehículo.
2. El sistema muestra la pantalla de modificación de vehículos con el vehículo seleccionado.
3. El usuario realiza las modificaciones deseadas.
4. El usuario pulsa sobre el botón guardar.
5. El sistema valida los datos.
6. El sistema guarda las modificaciones realizadas sobre el vehículo.
7. El sistema redirige al usuario a la pantalla de selección de vehículos.

Curso alternativo

1-4a. El usuario cancela el proceso.

1. El sistema no realiza ningún cambio.
2. El usuario es redirigido al caso de uso «Buscar Vehículo»

5a. Falta algún dato.

1. El sistema muestra el mensaje de error «Algún campo se encuentra vacío»
2. El sistema marca en rojo el campo vacío.

5b. Algún dato es incorrecto

1. El sistema muestra el mensaje de error «Error al validar los datos»
2. El sistema marca en rojo el campo.

3.2.2.2.4 CU Ver Vehículo

Caso de uso:	**Ver Vehículo**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado. - El usuario ha hecho login en la aplicación.
Postcondición:	- El vehículo es visualizado por el usuario.

Curso normal

1. El usuario desea ver un vehículo.
2. El sistema muestra un listado con sus vehículos.
3. El usuario selecciona el vehículo deseado.
4. El sistema muestra la pantalla con el vehículo seleccionado.
5. El vehículo es visualizado por el usuario.

Curso alternativo

1-3a. El usuario cancela el proceso

1. El sistema muestra la pantalla principal de la aplicación.

2a. La búsqueda no devuelve ningún resultado.

1. El sistema muestra un listado vacío con el mensaje «No se han encontrado vehículos con el filtro seleccionado»

2b. El usuario no se encuentra logueado/registrado.

1. El sistema muestra la pantalla de login con el mensaje «Necesita estar logueado para poder ver sus vehículos»

3.2.2.2.5 CU Eliminar Vehículo

Caso de uso:	**Eliminar Vehículo**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado. - El usuario ha hecho login en la aplicación.
Postcondición:	- El vehículo es eliminado.

Curso normal

1. El usuario desea eliminar un vehículo.
2. El sistema muestra un listado con sus vehículos.
3. El usuario filtra por los campos requeridos.
4. El usuario obtiene el resultado de su búsqueda.
5. El usuario selecciona el vehículo buscado y selecciona eliminar.
6. El sistema pide confirmación para eliminar el vehículo.
7. El sistema recuerda que se perderán todos los trayectos, mantenimientos y cualquier tipo de dato asociado y vuelve a pedir confirmación.
8. El vehículo es eliminado del sistema.

Curso alternativo

5a. La búsqueda no devuelve ningún resultado.

1. El sistema muestra un listado vacío con el mensaje «No se han encontrado vehículos con el filtro seleccionado»

3.2.2.2.6 CU Consultar listado de vehículos

Caso de uso:	**Consultar listado de vehiculos**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema.
Postcondición:	- El usuario obtiene el resultado de vehículos propios.

Curso normal

1. El usuario desea consultar el listado de sus vehículos.
2. El sistema muestra un listado con sus vehículos.

Curso alternativo

1-2a. El usuario cancela el proceso.

1. El sistema redirige al usuario a la pantalla principal.

3.2.2.3. CU Gestión de trayectos

3.2.2.3.1 CU Añadir Trayecto

Caso de uso:	**Añadir Trayecto**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene al menos un vehículo. - El usuario tiene un vehículo seleccionado por defecto.
Postcondición:	- Un nuevo trayecto será añadido al sistema y asociado al vehículo.

Curso normal

1. El usuario desea crear un nuevo trayecto.
2. El usuario presiona sobre el botón «Añadir trayecto».
3. El sistema valida que el usuario se encuentra logueado en el sistema, que tiene un vehículo seleccionado por defecto y no ha sobrepasado el límite de trayectos por vehículo.
4. El sistema muestra la pantalla de añadir trayecto configurada con los datos más probables (precio actual combustible, fecha inicio, fecha fin, etc)
5. El usuario introduce los datos del nuevo trayecto y presiona en guardar.
6. El sistema valida los datos introducidos.
7. El sistema guarda los datos del nuevo trayecto y los relaciona al vehículo.
8. El usuario es redirigido a la pantalla «Gestión de Trayectos».

Curso alternativo

1-5a. El usuario cancela el proceso.

1. El sistema cancela y redirecciona al listado de trayectos.

3a. El usuario no se encuentra logueado.

1. El sistema muestra el mensaje de error «Necesita estar logueado»
2. El sistema cancela y redirecciona a la pantalla de login.

3b. El usuario ha sobrepasado el límite de trayectos.

1. El sistema muestra el mensaje de error «Ha sobrepaso el límite de trayectos para este vehículo»

3c. El usuario no tiene vehículo seleccionado por defecto.

1. El sistema muestra el mensaje de error «Debe seleccionar un vehículo para añadir el trayecto»
2. El usuario es redirigido a la pantalla de selección de vehículo.

4a. No hay datos anteriores sobre los que realizar los cálculos necesarios.

1. Se muestra la pantalla sin ningún campo relleno por defecto.

5a. Falta algún dato por rellenar.

1. El sistema muestra el mensaje de error «Existen campos vacíos»
2. El sistema marca en rojo el campo.

5b. Algún dato es incorrecto

1. El sistema muestra el mensaje de error «Error al validar los datos»
2. El sistema marca en rojo el campo.

3.2.2.3.2 CU Buscar trayecto

Caso de uso:	**Buscar trayecto**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene al menos un vehículo. - El usuario tiene un vehículo seleccionado por defecto.
Postcondición:	- El usuario obtiene el resultado de su búsqueda.

Curso normal

1. El usuario desea buscar un trayecto.
2. El sistema muestra un listado con los trayectos del vehículo seleccionado por defecto.
3. El usuario filtra por los campos requeridos.
4. El usuario obtiene el resultado de su búsqueda.
5. El usuario selecciona el trayecto deseado.

Curso alternativo

1-5a. El usuario cancela el proceso.

1. El sistema redirige al usuario a la pantalla principal.

2a. El usuario no tiene ningún vehículo.

1. El sistema muestra el mensaje de error «No dispone de ningún vehículo, por favor añada uno nuevo»
2. El sistema redirige a la pantalla de creación de vehículo, caso de uso «Añadir vehículo».

2b. El usuario no tiene un vehículo seleccionado por defecto.

1. El sistema muestra el mensaje «Por favor, seleccione un vehículo».
2. El sistema muestra un combo para seleccionar el vehículo.

5. La búsqueda no devuelve ningún resultado.

1. El sistema muestra el mensaje «No se han encontrado trayectos que cumplan los requisitos».

3.2.2.3.3 CU Modificar Trayecto

Caso de uso:	**Modificar Trayecto**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene al menos un trayecto.
Postcondición:	- El trayecto seleccionado será modificado.

Curso normal

1. El usuario desea modificar un trayecto.
2. El sistema valida que el usuario se encuentra logueado en el sistema y que es el dueño.
3. El sistema muestra la pantalla de modificar trayecto.
4. El usuario introduce modifica los datos deseados.
5. El usuario pulsa le botón «Guardar».
6. El sistema valida los datos introducidos.
7. El sistema guarda los datos del trayecto.
8. El usuario es redirigido a la pantalla «Gestión de Trayectos».

Curso alternativo

1-5a. El usuario cancela el proceso.

1. El sistema cancela y redirecciona al listado de trayectos.

2a. El usuario no se encuentra logueado.

1. El sistema muestra el mensaje de error «Necesita estar logueado»
2. El sistema cancela y redirecciona a la pantalla de login.

2b. El usuario no es el dueño.

1. El sistema muestra el mensaje de error «No puede realizar esta acción».
2. El sistema debe registrar un error de seguridad ya que nunca debe ser posible esta acción.
3. El usuario es redirigido a la pantalla principal.

6a. Falta algún dato por rellenar.

1. El sistema muestra el mensaje de error «Existen campos vacíos»
2. El sistema marca en rojo el campo.

6b. Algún dato es incorrecto

1. El sistema muestra el mensaje de error «Error al validar los datos»
2. El sistema marca en rojo el campo.

3.2.2.3.4 CU Ver Trayecto

Caso de uso:	**Ver Trayecto**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene al menos un trayecto.
Postcondición:	- El trayecto seleccionado será visualizado por el usuario.

Curso normal

1. El usuario desea ver un trayecto.
2. El sistema valida que el usuario se encuentra logueado en el sistema y que es el dueño.
3. El sistema muestra la pantalla de ver trayecto.

Curso alternativo

2a. El usuario no se encuentra logueado.

1. El sistema muestra el mensaje de error «Necesita estar logueado»
2. El sistema cancela y redirecciona a la pantalla de login.

2b. El usuario no es el dueño.

1. El sistema muestra el mensaje de error «No puede realizar esta acción».
2. El sistema debe registrar un error de seguridad ya que nunca debe ser posible esta acción.
3. El usuario es redirigido a la pantalla principal.

3.2.2.3.5 CU Eliminar Trayecto

Caso de uso:	**Eliminar trayecto**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene un coche seleccionado por defecto. - El usuario tiene al menos un trayecto.
Postcondición:	- El trayecto es eliminado.

Curso normal

1. El usuario desea eliminar un trayecto.
2. El sistema pide confirmación para eliminar el trayecto.
3. El trayecto es eliminado del sistema.

Curso alternativo

1-2a. El usuario cancela el proceso.

1. El sistema cancela la eliminación del trayecto.
2. El sistema redirige al usuario al listado de trayectos.

2a. El usuario no confirma.

1. El sistema cancela la eliminación del trayecto.
2. El sistema redirige al usuario al listado de trayectos.

3.2.2.3.6 CU Consultar listado de trayectos

Caso de uso:	**Consultar listado de trayectos**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene un coche seleccionado por defecto.
Postcondición:	- El usuario obtiene el resultado de trayectos del vehículo seleccionado.

Curso normal

1. El usuario desea consultar el listado de sus trayectos.
2. El sistema muestra un listado con los trayectos del vehículo seleccionado.

Curso alternativo

1-2a. El usuario cancela el proceso.

1. El sistema redirige al usuario a la pantalla principal.

3.2.2.4. CU Gestión de piezas

3.2.2.4.1 CU Añadir Pieza

Caso de uso:	**Añadir Pieza**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene al menos un vehículo. - El usuario tiene un vehículo seleccionado por defecto. - La pieza no ha sido añadida previamente al vehículo.
Postcondición:	- Una nueva pieza será añadida al vehículo.

Curso normal

1. El usuario desea añadir una nueva pieza.
2. El usuario presiona sobre el botón «Añadir pieza» en el listado de piezas del vehículo.
3. El sistema muestra la pantalla de añadir pieza.
4. El usuario rellena los datos de la nueva pieza a añadir.
5. El usuario pulsa sobre el botón guardar.
6. El sistema valida los datos.
7. El sistema añade la nueva pieza.
8. El sistema redirige al usuario al listado de piezas del vehículo.

Curso alternativo

1-5a. El usuario cancela el proceso.

1. El sistema cancela y redirecciona al listado de piezas.

2a. El usuario no se encuentra logueado.

1. El sistema muestra el mensaje de error «Necesita estar logueado»
2. El sistema cancela y redirecciona a la pantalla de login.

2b. El usuario no tiene vehículo seleccionado por defecto.

1. El sistema muestra el mensaje de error «Debe seleccionar un vehículo para añadir el trayecto»
2. El usuario es redirigido a la pantalla de selección de vehículo.

6a. El vehículo ya tiene la pieza

1. El sistema muestra el mensaje de error «La pieza seleccionada ya se encuentra añadida».

6b. Falta algún dato por rellenar.

1. El sistema muestra el mensaje de error «Existen campos vacíos»
2. El sistema marca en rojo el campo.

6c. Algún dato es incorrecto

1. El sistema muestra el mensaje de error «Error al validar los datos»
2. El sistema marca en rojo el campo.

3.2.2.4.2 CU Modificar Pieza

Caso de uso:	**Modificar trayecto**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene al menos un vehículo.
Postcondición:	- Los datos de la pieza son modificados.

Curso normal

1. El usuario desea modificar una pieza.
2. El usuario selecciona la pieza a modificar desde el listado.
3. El sistema muestra la pantalla de modificación de piezas.
4. El usuario realiza las modificaciones deseadas.
5. El usuario presiona sobre el botón «Guardar».
6. El sistema valida los datos.
7. El sistema guarda las modificaciones.
8. El sistema redirige al listado de piezas.

Curso alternativo

1-5a. El usuario cancela el proceso.

1. El sistema redirige al usuario a la pantalla principal.

2a. El usuario no tiene ningún vehículo.

1. El sistema muestra el mensaje de error «No dispone de ningún vehículo, por favor añada uno nuevo»
2. El sistema redirige a la pantalla de creación de vehículo, caso de uso «Añadir vehículo».

2b. El usuario no tiene un vehículo seleccionado por defecto.

1. El sistema muestra el mensaje «Por favor, seleccione un vehículo».
2. El sistema muestra un combo para seleccionar el vehículo.

3a. El vehículo no tiene ninguna pieza.

1. El sistema muestra el mensaje «No se han encontrado piezas en el vehículo seleccionado».

6a. Falta algún dato por rellenar.

1. El sistema muestra el mensaje de error «Existen campos vacíos»
2. El sistema marca en rojo el campo.

6b. Algún dato es incorrecto

1. El sistema muestra el mensaje de error «Error al validar los datos»
2. El sistema marca en rojo el campo.

3.2.2.4.3 CU Ver Pieza

Caso de uso:	**Ver Pieza**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene un vehículo seleccionado. - El vehículo tiene al menos una pieza.
Postcondición:	- El usuario visualiza la pieza seleccionada.

Curso normal

1. El usuario desea visualizar un trayecto.
2. El usuario selecciona la pieza a visualizar desde el listado de piezas.
3. El sistema muestra la pantalla con la pieza seleccionada.

Curso alternativo

1-2a. El usuario cancela el proceso.

1. El sistema cancela y redirecciona al listado de piezas.

2a. El usuario no se encuentra logueado.

1. El sistema muestra el mensaje de error «Necesita estar logueado»
2. El sistema cancela y redirecciona a la pantalla de login.

3.2.2.4.4 CU Eliminar Pieza

Caso de uso:	**Eliminar pieza**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene un coche seleccionado por defecto. - El vehículo tiene al menos una pieza.
Postcondición:	- La pieza es eliminada del vehículo.

Curso normal

1. El usuario desea eliminar una pieza.
2. El usuario selecciona la pieza a eliminar.
3. El sistema pide confirmación antes de eliminar la pieza.
4. El sistema elimina la pieza.
5. El sistema redirige al usuario al listado de piezas.

Curso alternativo

1-3a. El usuario cancela el proceso.

1. El sistema cancela la eliminación de la pieza.
2. El sistema redirige al usuario al listado de piezas.

2a. El usuario no tiene ningún vehículo.

1. El sistema muestra el mensaje de error «No dispone de ningún vehículo, por favor añada uno nuevo»
2. El sistema redirige a la pantalla de creación de vehículo, caso de uso «Añadir vehículo».

2b. El vehículo no tiene ninguna pieza.

1. El sistema muestra el mensaje «No se ha encontrado ninguna pieza».

3a. El usuario no confirma la operación.

1. El sistema cancela la eliminación de la pieza.
2. El sistema redirige a la pantalla con el listado de piezas.

3.2.2.4.5 CU Consultar listado de piezas

Caso de uso:	**Consultar listado de piezas**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene un coche seleccionado por defecto.
Postcondición:	- El usuario obtiene el resultado con las piezas del vehículo seleccionado.

Curso normal

1. El usuario desea consultar el listado de sus piezas.
2. El sistema muestra un listado con las piezas del vehículo seleccionado.

Curso alternativo

1-2a. El usuario cancela el proceso.

1. El sistema redirige al usuario a la pantalla principal.

2a. El usuario no tiene ningún vehículo.

1. El sistema muestra el mensaje de error «No dispone de ningún vehículo, por favor añada uno nuevo»
2. El sistema redirige a la pantalla de creación de vehículo, caso de uso «Añadir vehículo».

2b. El vehículo no tiene ninguna pieza.

1. El sistema muestra el mensaje siguiente en lugar del listdo «El vehículo seleccionado no tiene ninguna pieza»

3.2.2.4.6 CU Sustituir Pieza

Caso de uso:	**Sustituir pieza**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene al menos un vehículo. - El vehículo seleccionado tiene al menos una pieza.
Postcondición:	- Se registra un mantenimiento en una pieza determinada.

Curso normal

1. El usuario desea registrar el mantenimiento a una pieza determinada.
2. El usuario selecciona la pieza a sustituir.
3. El sistema muestra la pantalla de sustitución de pieza.
4. El usuario registra los datos de la sustitución.
5. El usuario pulsa sobre el botón «Sustituir».
6. El sistema valida los datos.
7. El sistema realiza la sustitución de la pieza, añadiendo al anterior al historial y inicializando la vida de la sustituida.
8. El sistema muestra la pantalla «Ver Pieza».

Curso alternativo

1-5a. El usuario cancela el proceso.

1. El sistema redirige al usuario a la pantalla principal.

2a. El usuario no tiene ningún vehículo.

1. El sistema muestra el mensaje de error «No dispone de ningún vehículo, por favor añada uno nuevo».
2. El sistema redirige a la pantalla de creación de vehículo, caso de uso «Añadir vehículo».

2b. El usuario no tiene un vehículo seleccionado por defecto.

1. El sistema muestra el mensaje «Por favor, seleccione un vehículo».
2. El sistema muestra un combo para seleccionar el vehículo.

3a. El vehículo no tiene ninguna pieza.

1. El sistema muestra el mensaje «No se han encontrado piezas en el vehículo seleccionado».

6a. Falta algún dato por rellenar.

1. El sistema muestra el mensaje de error «Existen campos vacíos»
2. El sistema marca en rojo el campo.

6b. Algún dato es incorrecto

1. El sistema muestra el mensaje de error «Error al validar los datos»
2. El sistema marca en rojo el campo.

3.2.2.4.7 CU Buscar Pieza

Caso de uso:	**Buscar pieza**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene al menos un vehículo.
Postcondición:	- El usuario obtiene el resultado de la búsqueda.

Curso normal

1. El usuario desea buscar una pieza.
2. El usuario visualiza el listado completo de piezas.
3. El usuario afina la búsqueda haciendo uso de los filtros.
4. El usuario obtiene el resultado de la búsqueda.

Curso alternativo

1-4a. El usuario cancela el proceso.

1. El sistema redirige al usuario a la pantalla principal.

2a. El usuario no tiene ningún vehículo.

1. El sistema muestra el mensaje de error «No dispone de ningún vehículo, por favor añada uno nuevo»
2. El sistema redirige a la pantalla de creación de vehículo, caso de uso «Añadir vehículo».

2b. El usuario no tiene un vehículo seleccionado por defecto.

1. El sistema muestra el mensaje «Por favor, seleccione un vehículo».
2. El sistema muestra un combo para seleccionar el vehículo.

2c. El vehículo no tiene ninguna pieza.

1. El sistema muestra el mensaje «No se han encontrado piezas en el vehículo seleccionado».

3.2.2.5. CU Consultar estadísticas

3.2.2.5.1 CU Ver estadísticas coche

Caso de uso:	**Ver estadísticas vehículos**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene al menos un vehículo.
Postcondición:	- El usuario visualiza las estadísticas.

Curso normal

1. El usuario desea ver las estadísticas de sus vehículos.
2. El sistema valida que el usuario se encuentra logueado en el sistema y tiene al menos un vehículo.
3. El sistema muestra la pantalla con los datos deseados.

Curso alternativo

2a. El usuario no se encuentra logueado.

1. El sistema muestra el mensaje de error «Necesita estar logueado»
2. El sistema cancela y redirecciona al listado de vehículos.

2b. El usuario no tiene ningún vehículo.

1. El sistema muestra el mensaje de error «Necesita crear un vehículo»
2. El sistema muestra la pantalla «Añadir vehiculo»

3.2.2.5.2 CU Ver estadísticas combustible

Caso de uso:	**Ver estadísticas combustibles**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene al menos un vehículo. - El usuario a registrado al menos un trayecto en alguno de sus vehículos.
Postcondición:	- El usuario visualiza las estadísticas.

Curso normal

1. El usuario desea ver las estadísticas de evolución del precio del combustible.
2. El sistema valida que el usuario se encuentra logueado en el sistema, tiene al menos un vehículo y al menos un trayecto.
3. El sistema muestra la pantalla con los datos deseados.

Curso alternativo

2a. El usuario no se encuentra logueado.

1. El sistema muestra el mensaje de error «Necesita estar logueado»
2. El sistema cancela y redirecciona al listado de vehículos.

2b. El usuario no tiene ningún vehículo.

1. El sistema muestra el mensaje de error «Necesita crear un vehículo»
2. El sistema muestra la pantalla «Añadir vehiculo»

2c. El usuario no ha registrado ningún trayecto.

1. El sistema muestra el mensaje de error «Necesita crear un trayecto en alguno de sus vehículos»

3.2.2.5.3 CU Ver listado de gráficas disponibles

Caso de uso:	**Ver estadísticas disponibles**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema.
Postcondición:	- El usuario visualiza las estadísticas.

Curso normal

1. El usuario desea ver el listado con las gráficas disponibles.
2. El sistema muestra la pantalla con el listado de gráficas.

3.2.2.5.4 CU Ver estadísticas distancias

Caso de uso:	**Ver estadísticas distancias**
Actores:	- Usuario
Precondición:	- El usuario se encuentra registrado y logueado en el sistema. - El usuario tiene al menos un vehículo. - El usuario a registrado al menos un trayecto en alguno de sus vehículos.
Postcondición:	- El usuario visualiza las estadísticas.

Curso normal

1. El usuario desea ver las estadísticas de kms realizados.
2. El sistema valida que el usuario se encuentra logueado en el sistema, tiene al menos un vehículo y al menos un trayecto.
3. El sistema muestra la pantalla con los datos deseados.

Curso alternativo

2a. El usuario no se encuentra logueado.

1. El sistema muestra el mensaje de error «Necesita estar logueado»
2. El sistema cancela y redirecciona al listado de vehículos.

2b. El usuario no tiene ningún vehículo.

1. El sistema muestra el mensaje de error «Necesita crear un vehículo»
2. El sistema muestra la pantalla «Añadir vehiculo»

2c. El usuario no ha registrado ningún trayecto.

1. El sistema muestra el mensaje de error «Necesita crear un trayecto en alguno de sus vehículos»

3.2.3. Diagramas de Interacción

3.2.3.1. Diagramas CU Gestión Usuarios

3.2.3.1.1 CU Baja Usuario

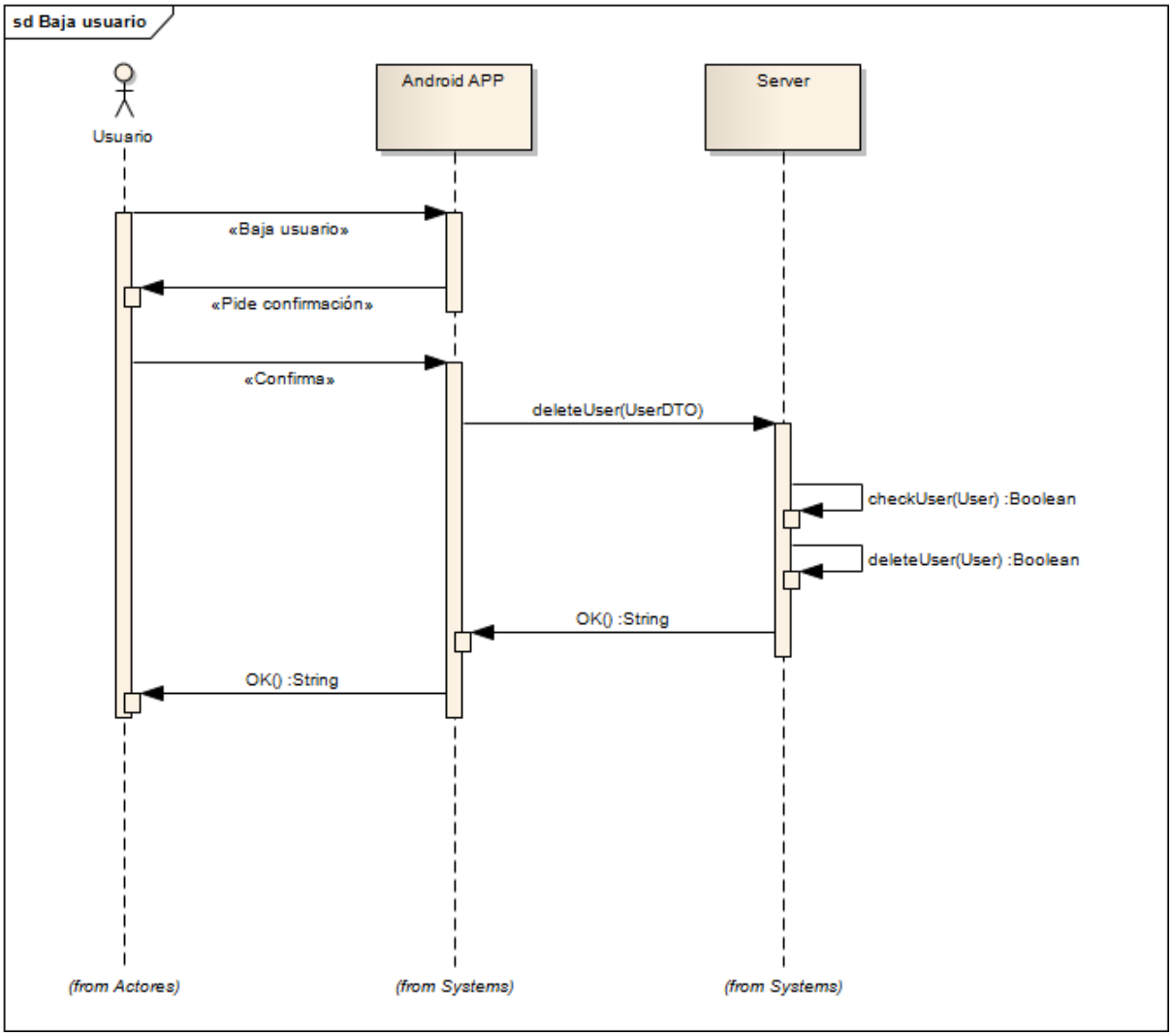

Figura 3.7: Diagrama de Interacción: Caso de Uso «Baja Usuario»

3.2.3.1.2 CU Login Usuario

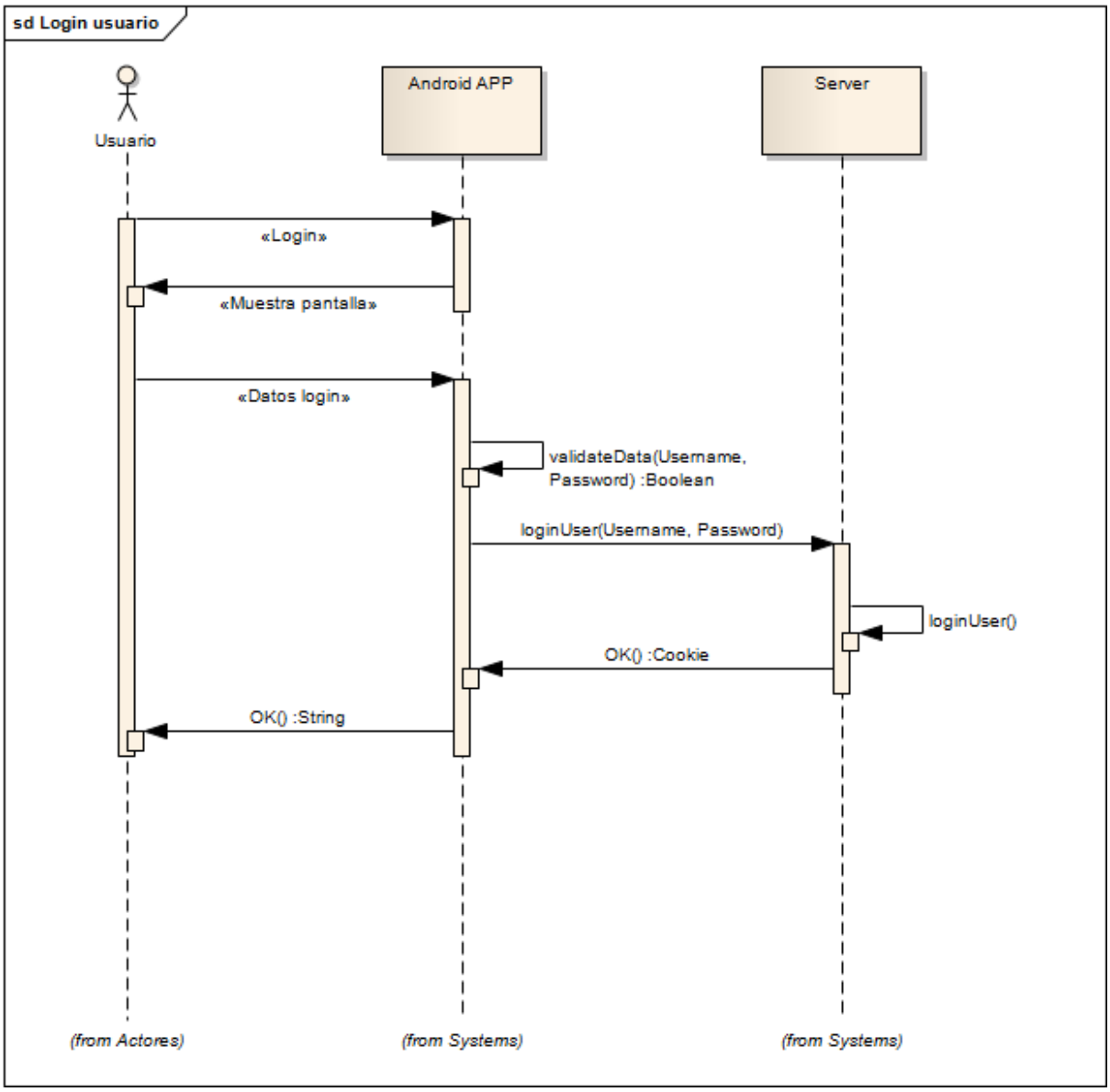

Figura 3.8: Diagrama de Interacción: Caso de Uso «Login Usuario»

3.2.3.1.3 CU Registro Usuario

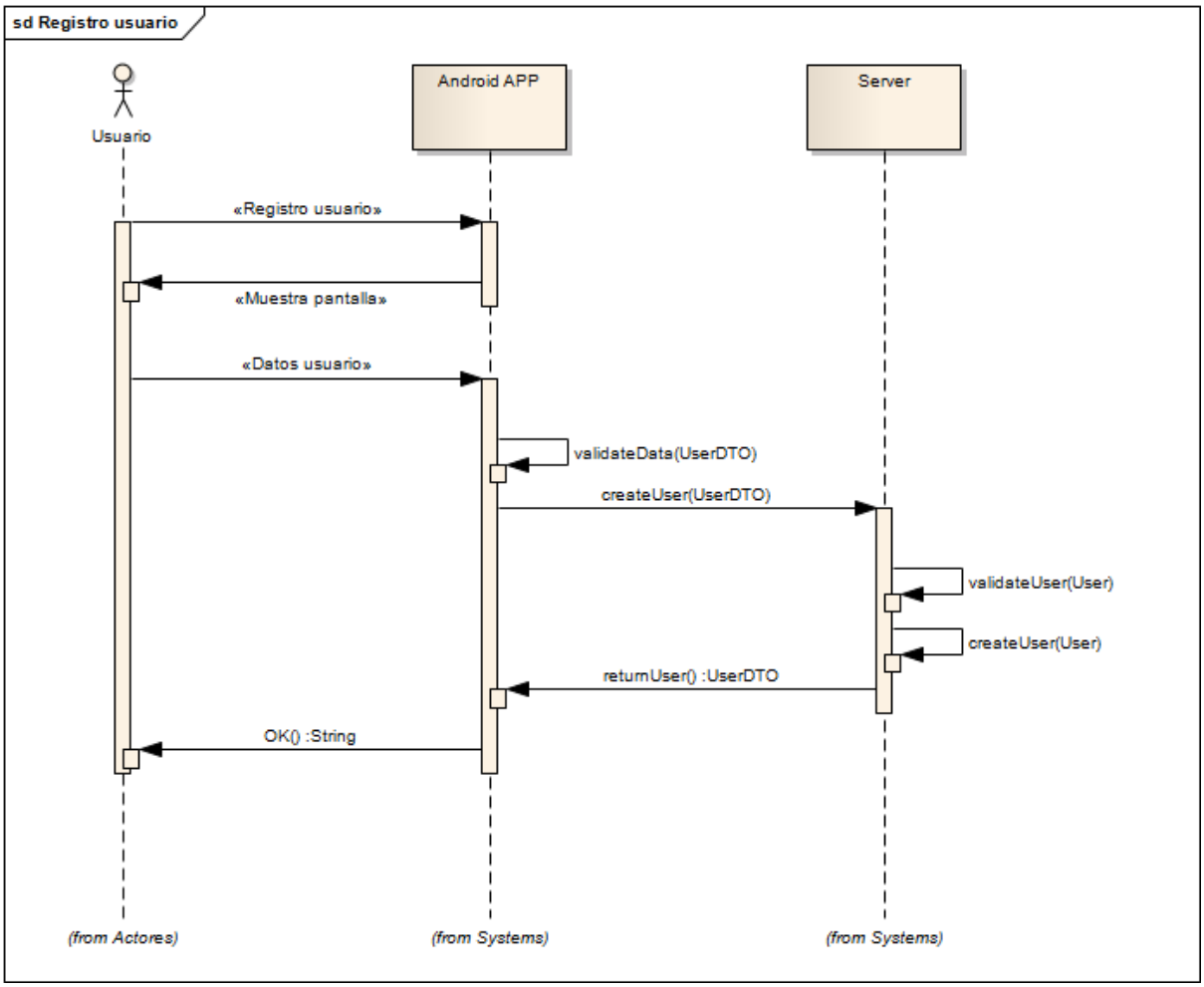

Figura 3.9: Diagrama de Interacción: Caso de Uso «Registro Usuarios»

3.2.3.2. Diagramas CU Gestión de vehículos

3.2.3.2.1 CU Añadir Vehículo

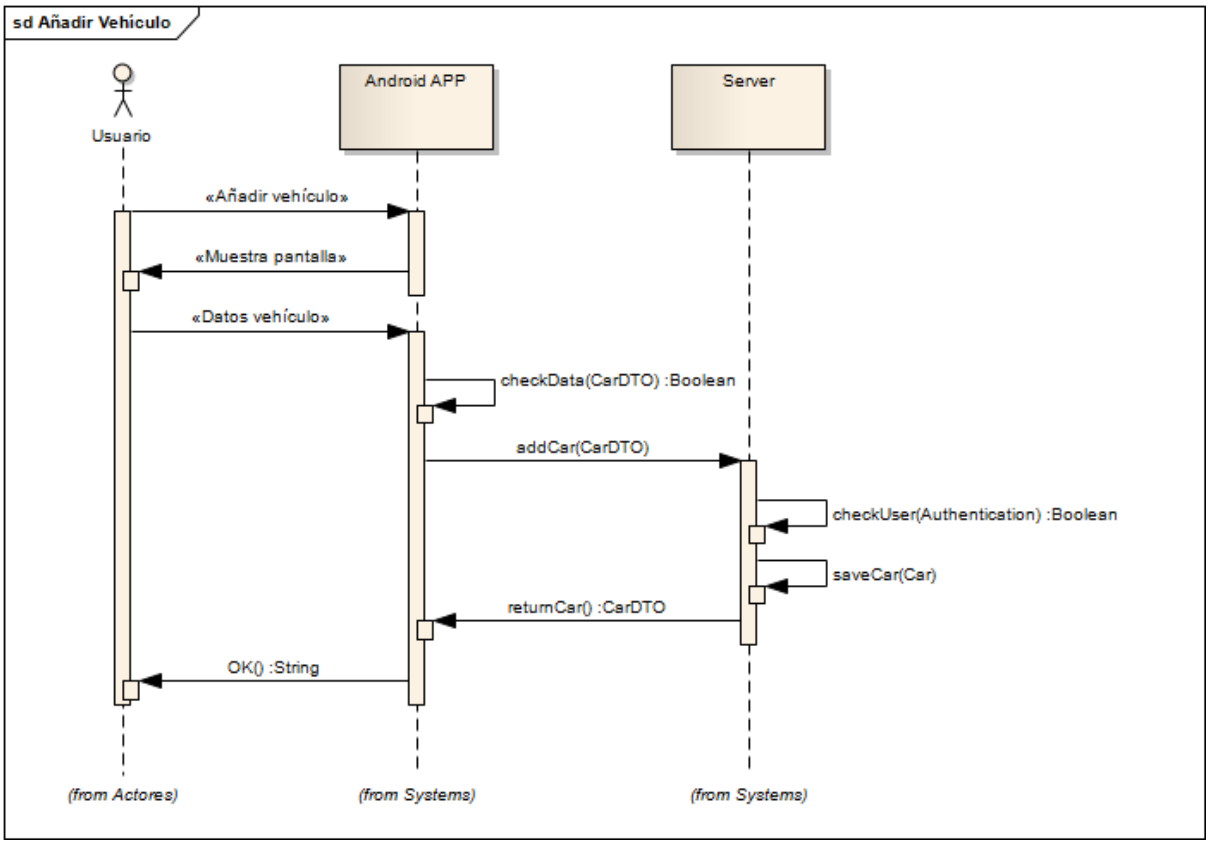

Figura 3.10: Diagrama de Interacción: Caso de Uso «Añadir Vehículo»

3.2.3.2.2 CU Buscar Vehículo

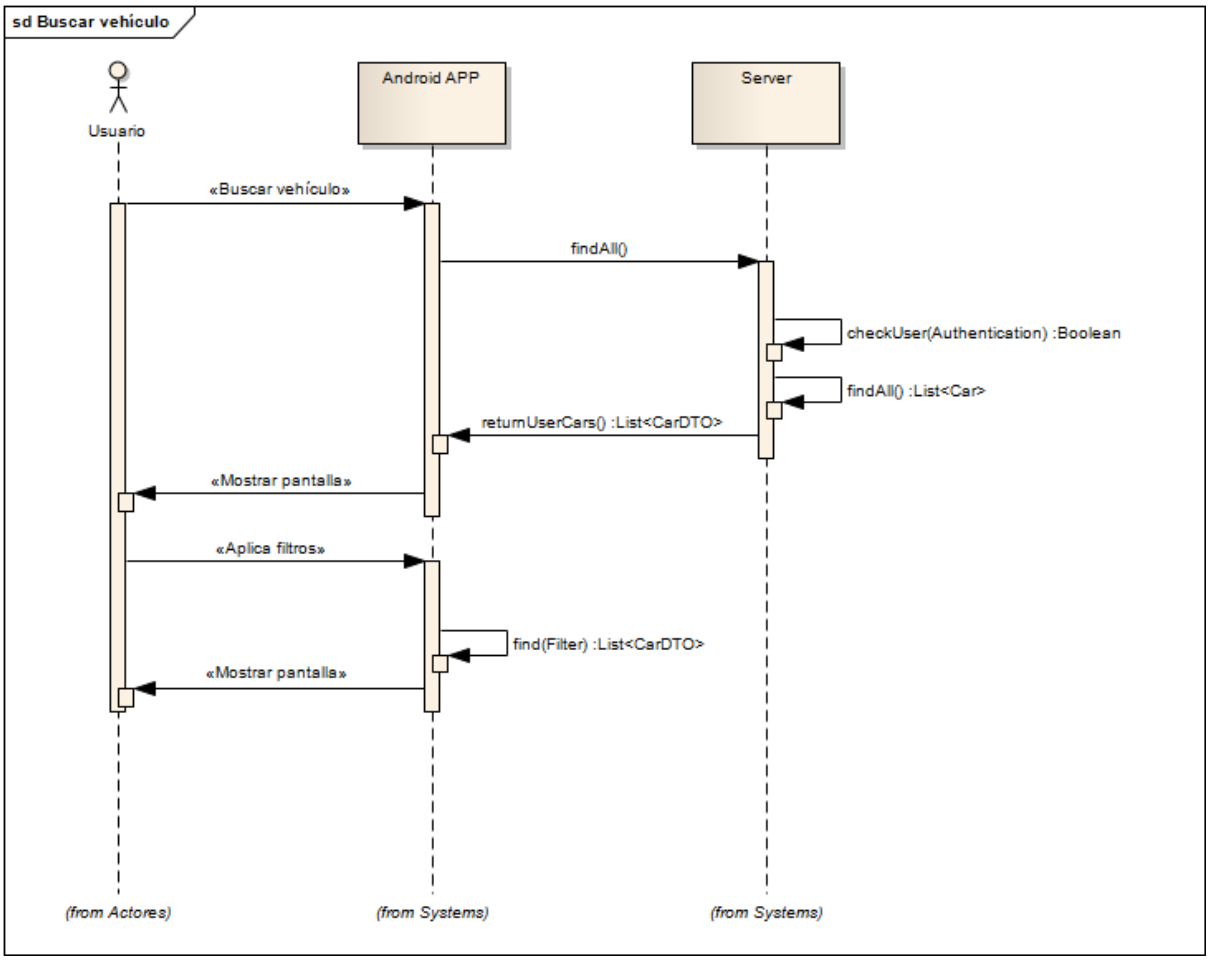

Figura 3.11: Diagrama de Interacción: Caso de Uso «Buscar Vehículo»

3.2.3.2.3 CU Modificar Vehículo

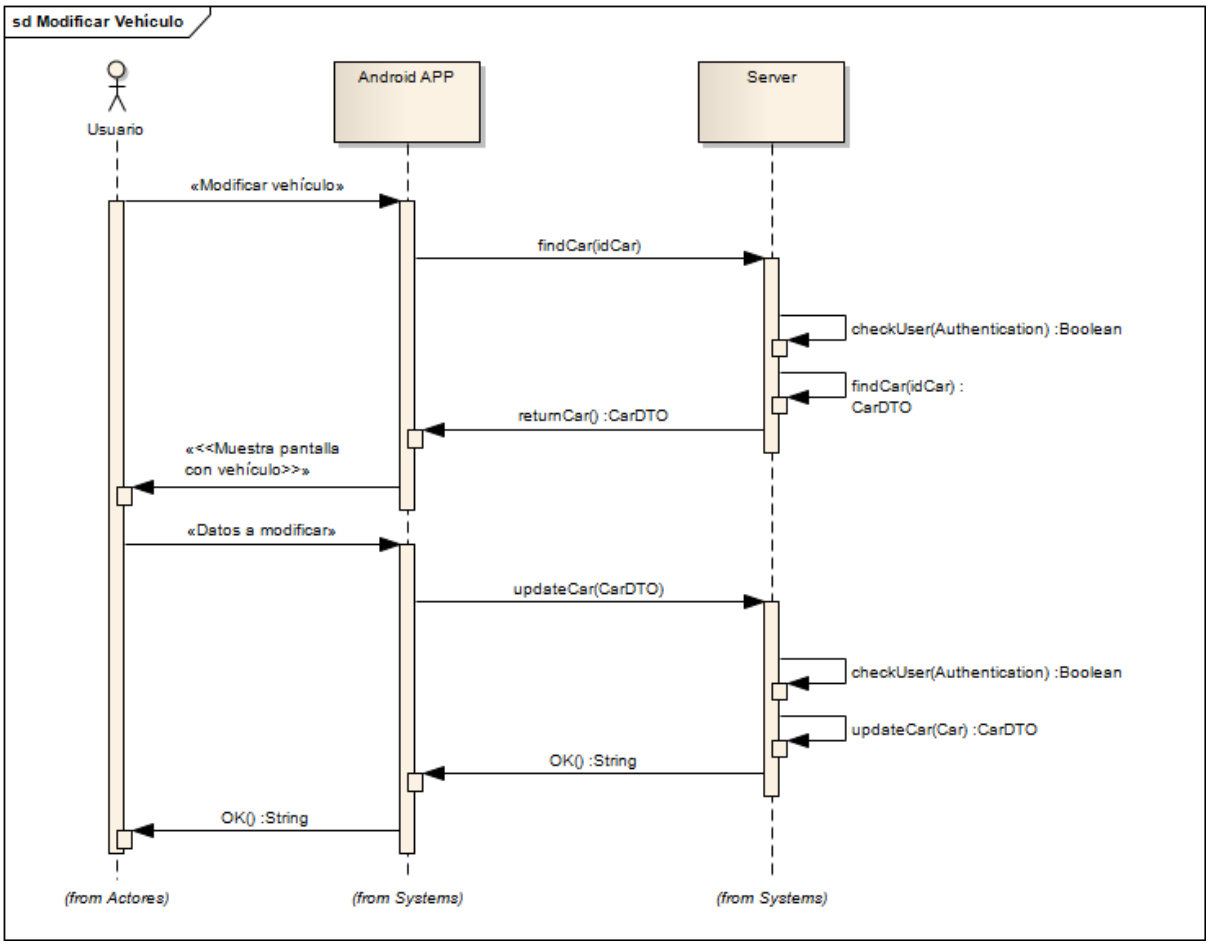

Figura 3.12: Diagrama de Interacción: Caso de Uso «Modificar Vehículo»

3.2.3.2.4 CU Eliminar Vehículo

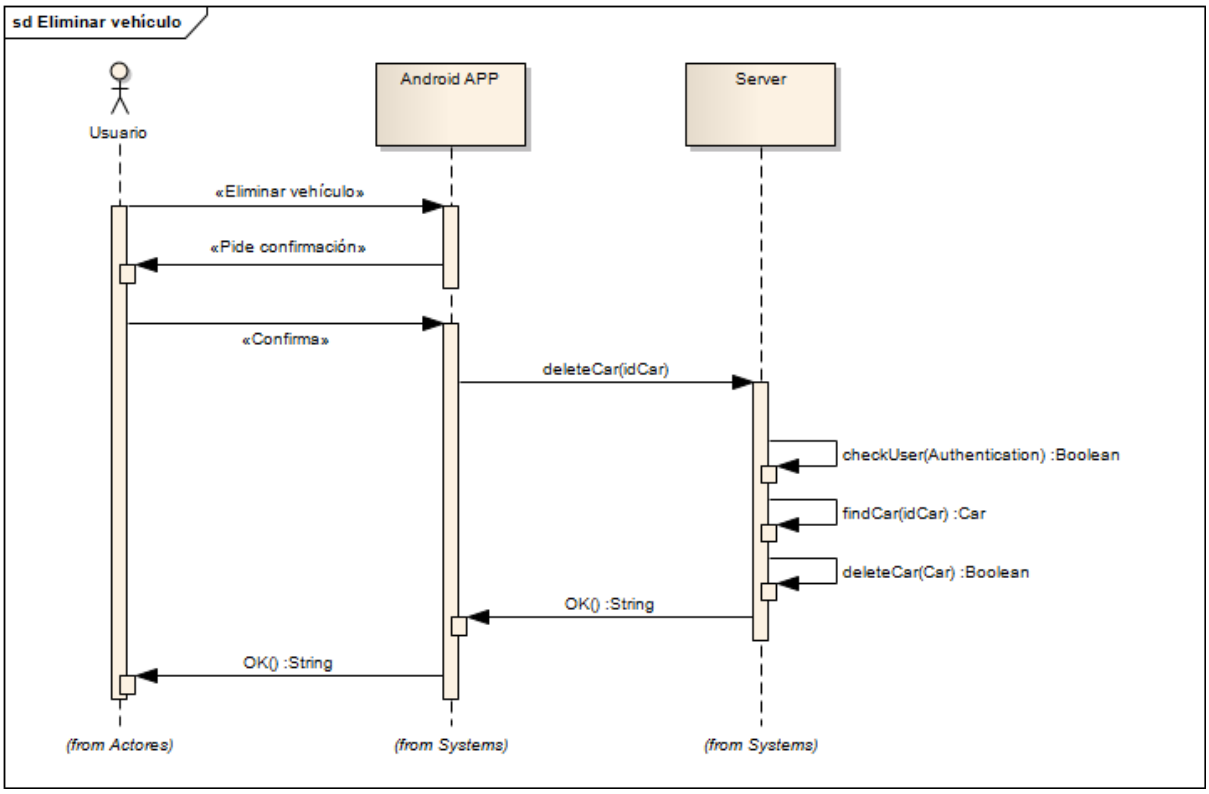

Figura 3.13: Diagrama de Interacción: Caso de Uso «Eliminar Vehículo»

3.2.3.2.5 CU Consultar listado de vehículos

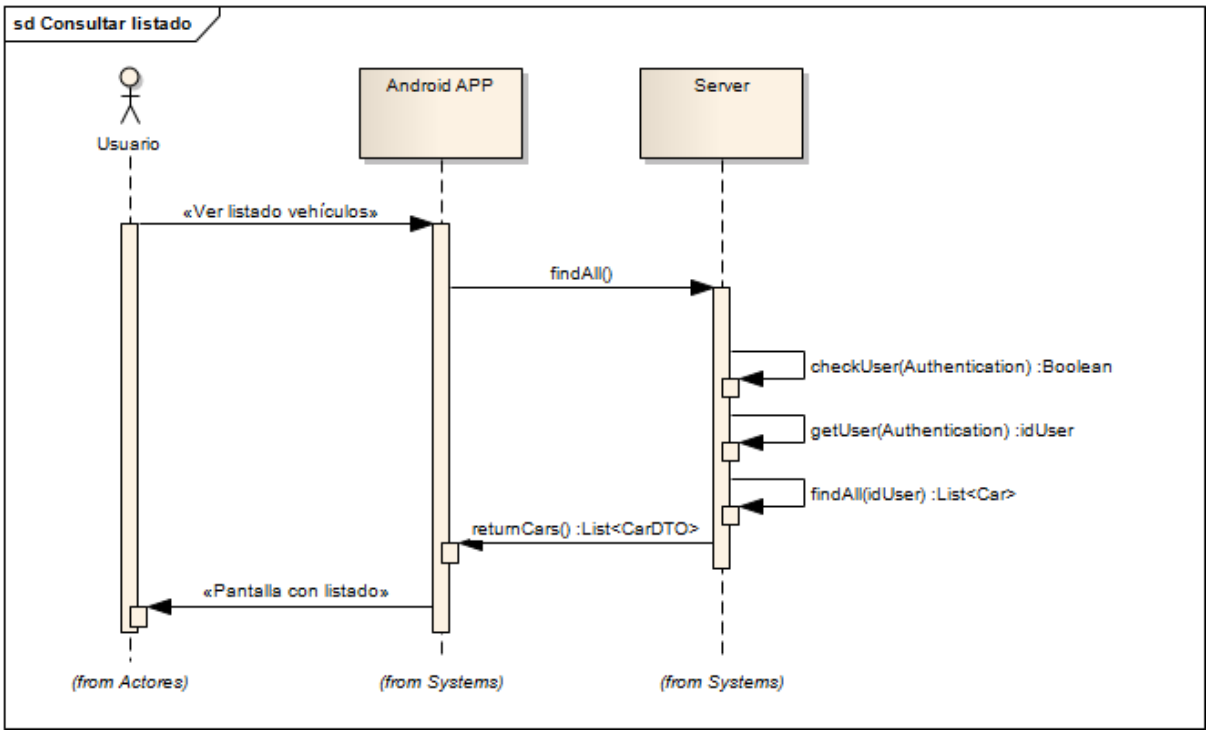

Figura 3.14: Diagrama de Interacción: Caso de Uso «Ver listado de vehículos»

3.2.3.2.6 CU Ver vehículo

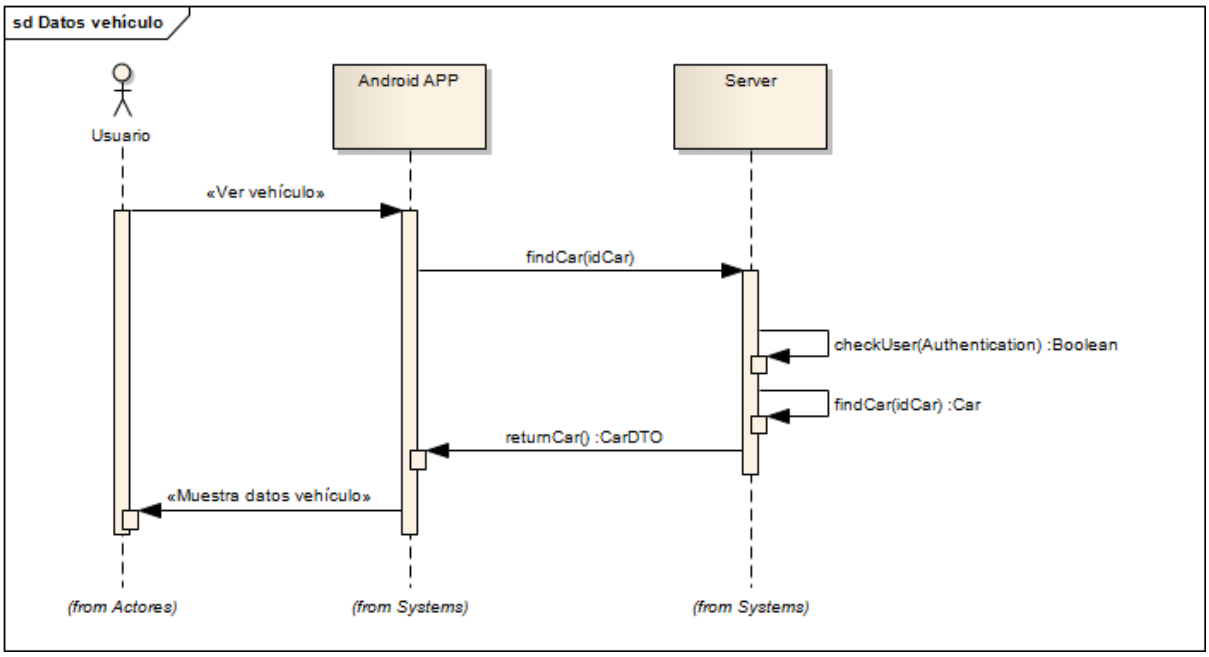

Figura 3.15: Diagrama de Interacción: Caso de Uso «Ver vehículo»

3.2.3.3. Diagramas CU Gestión de trayectos

3.2.3.3.1 CU Añadir Trayecto

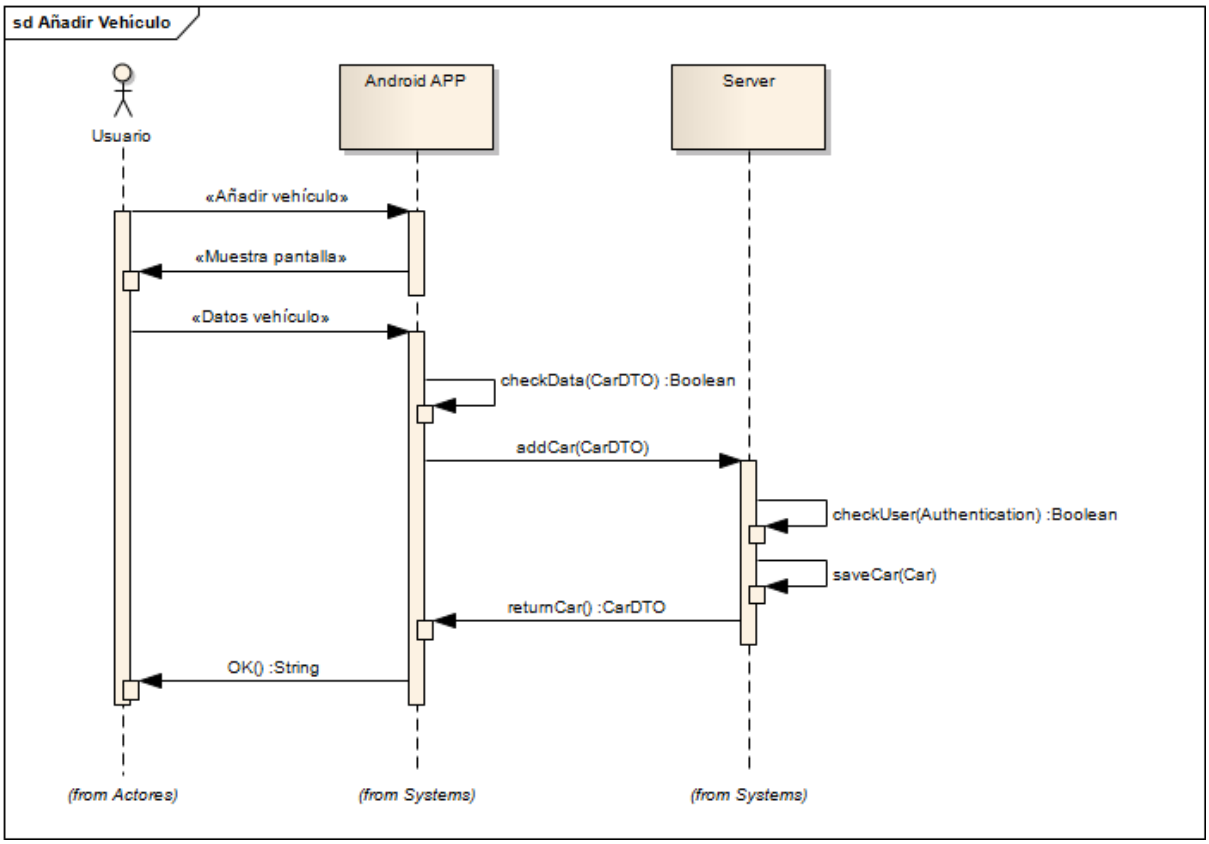

Figura 3.16: Diagrama de Interacción: Caso de Uso «Añadir Vehículo»

3.2.3.3.2 CU Buscar trayecto

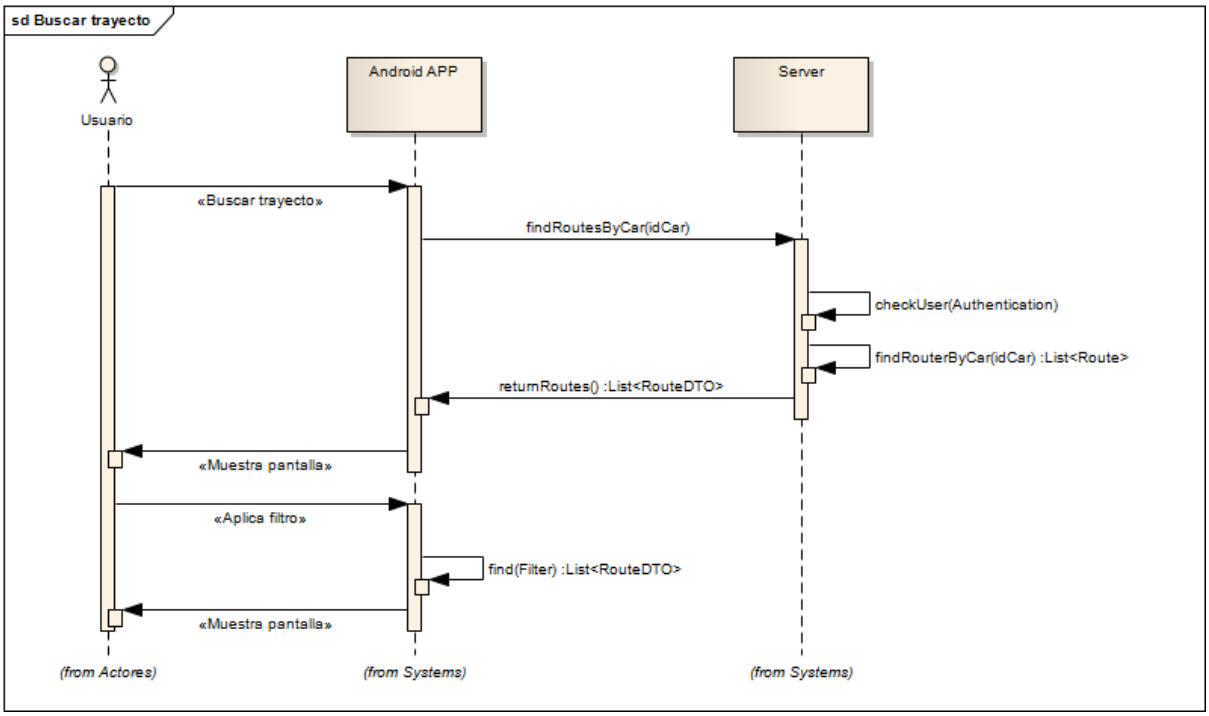

Figura 3.17: Diagrama de Interacción: Caso de Uso «Buscar Trayecto»

3.2.3.3.3 CU Modificar Trayecto

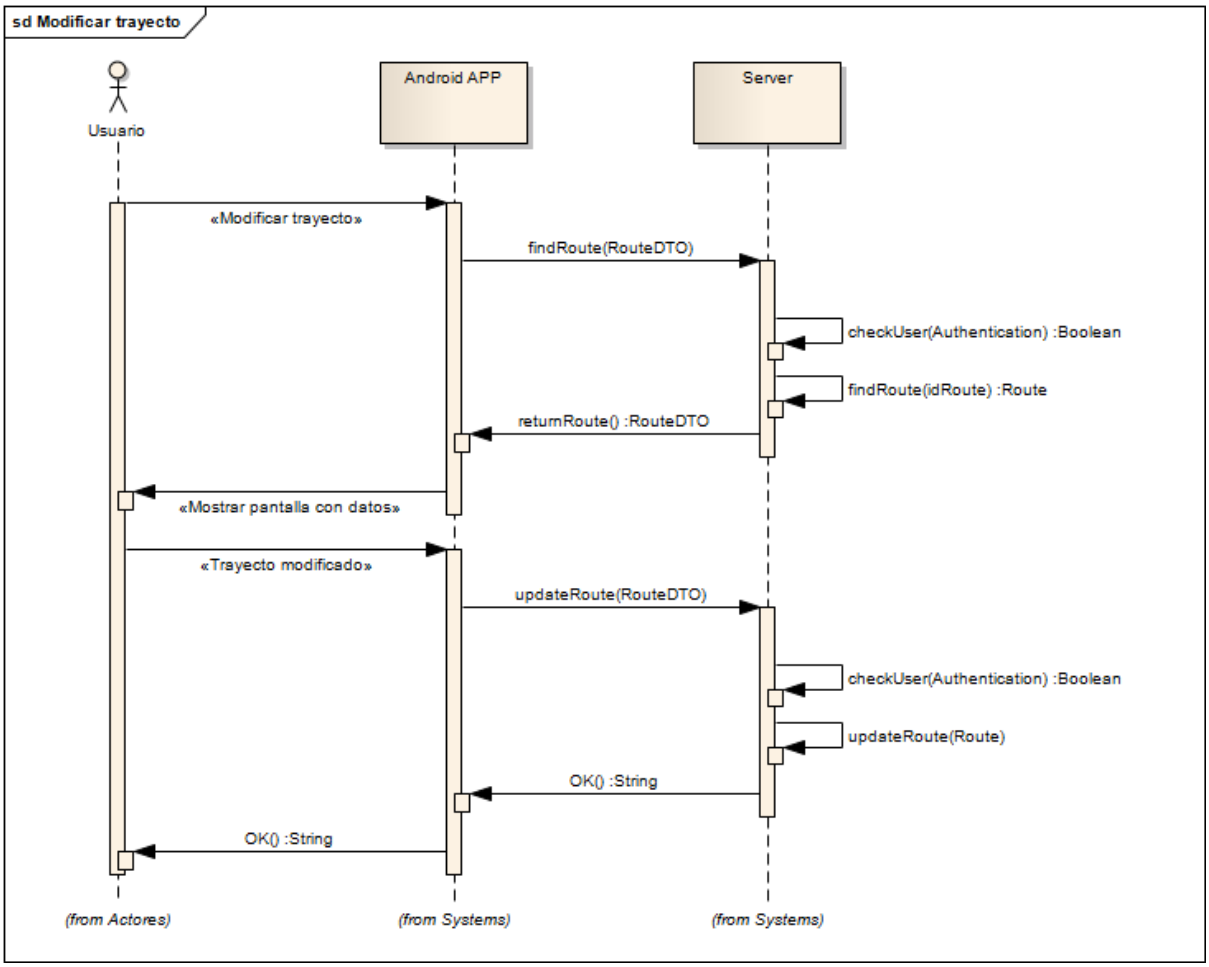

Figura 3.18: Diagrama de Interacción: Caso de Uso «Modificar Trayecto»

3.2.3.3.4 CU Eliminar Trayecto

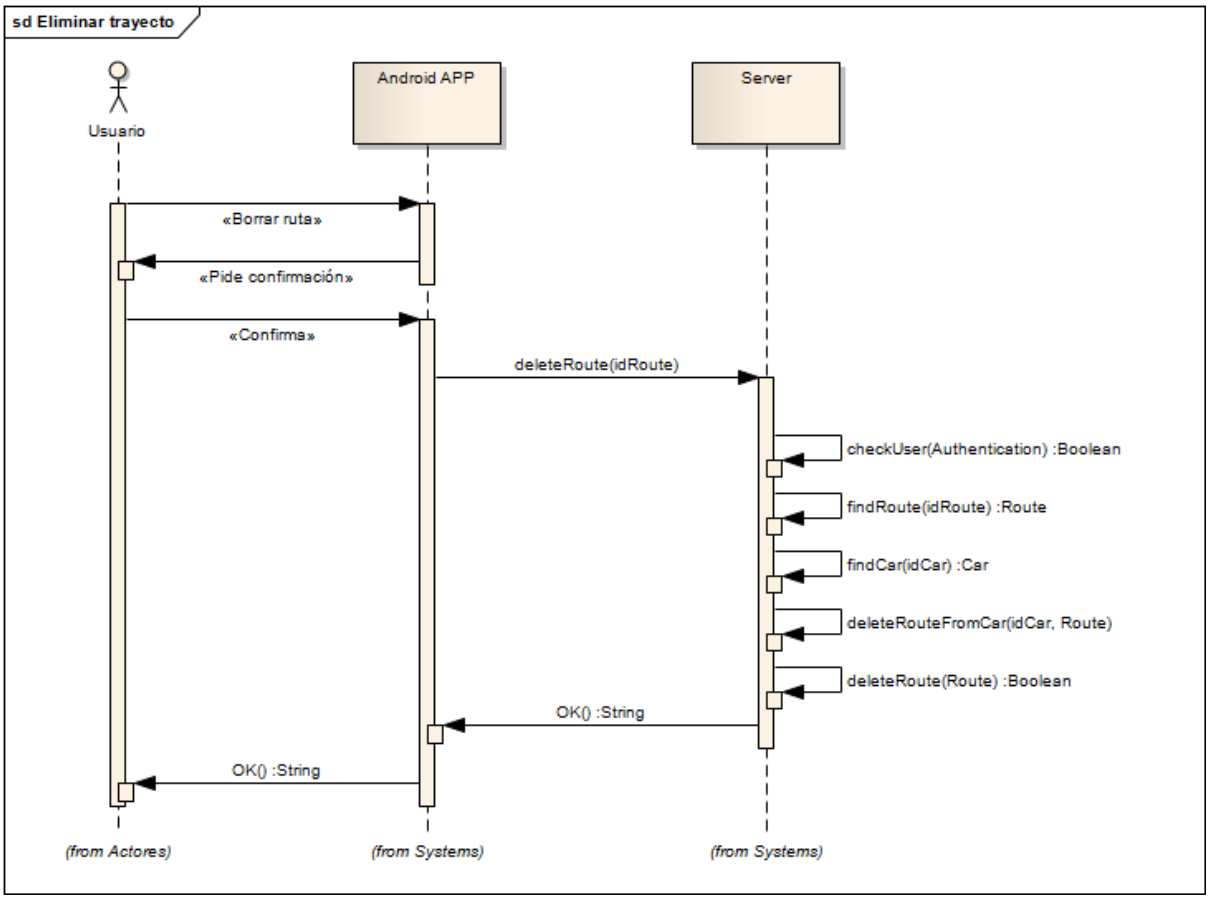

Figura 3.19: Diagrama de Interacción: Caso de Uso «Eliminar Trayecto»

3.2.3.3.5 CU Consultar listado de trayectos

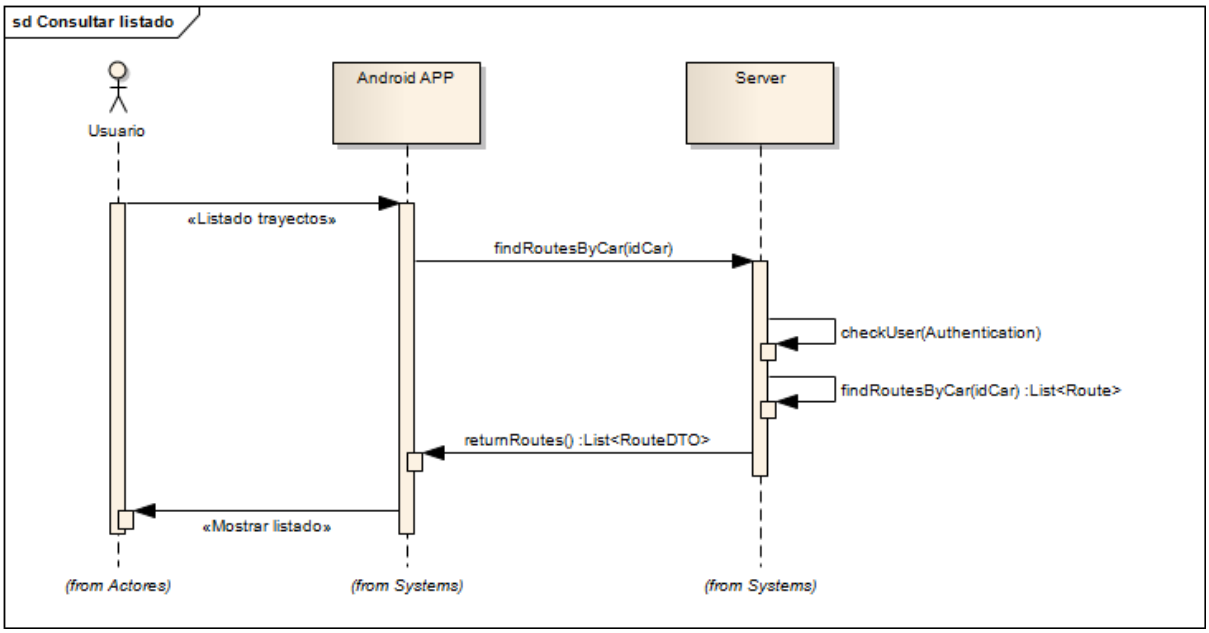

Figura 3.20: Diagrama de Interacción: Caso de Uso «Consultar listado de trayectos»

3.2.3.3.6 CU Ver trayecto

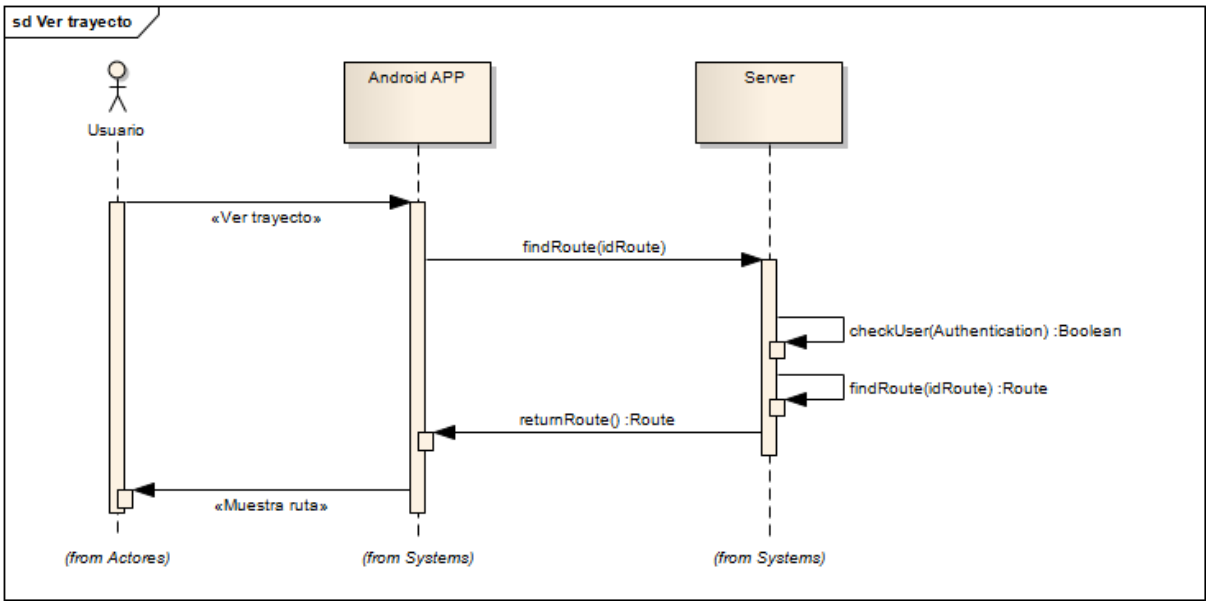

Figura 3.21: Diagrama de Interacción: Caso de Uso «Ver trayecto»

3.2.3.4. Diagramas CU Gestión de piezas

3.2.3.4.1 CU Añadir Pieza

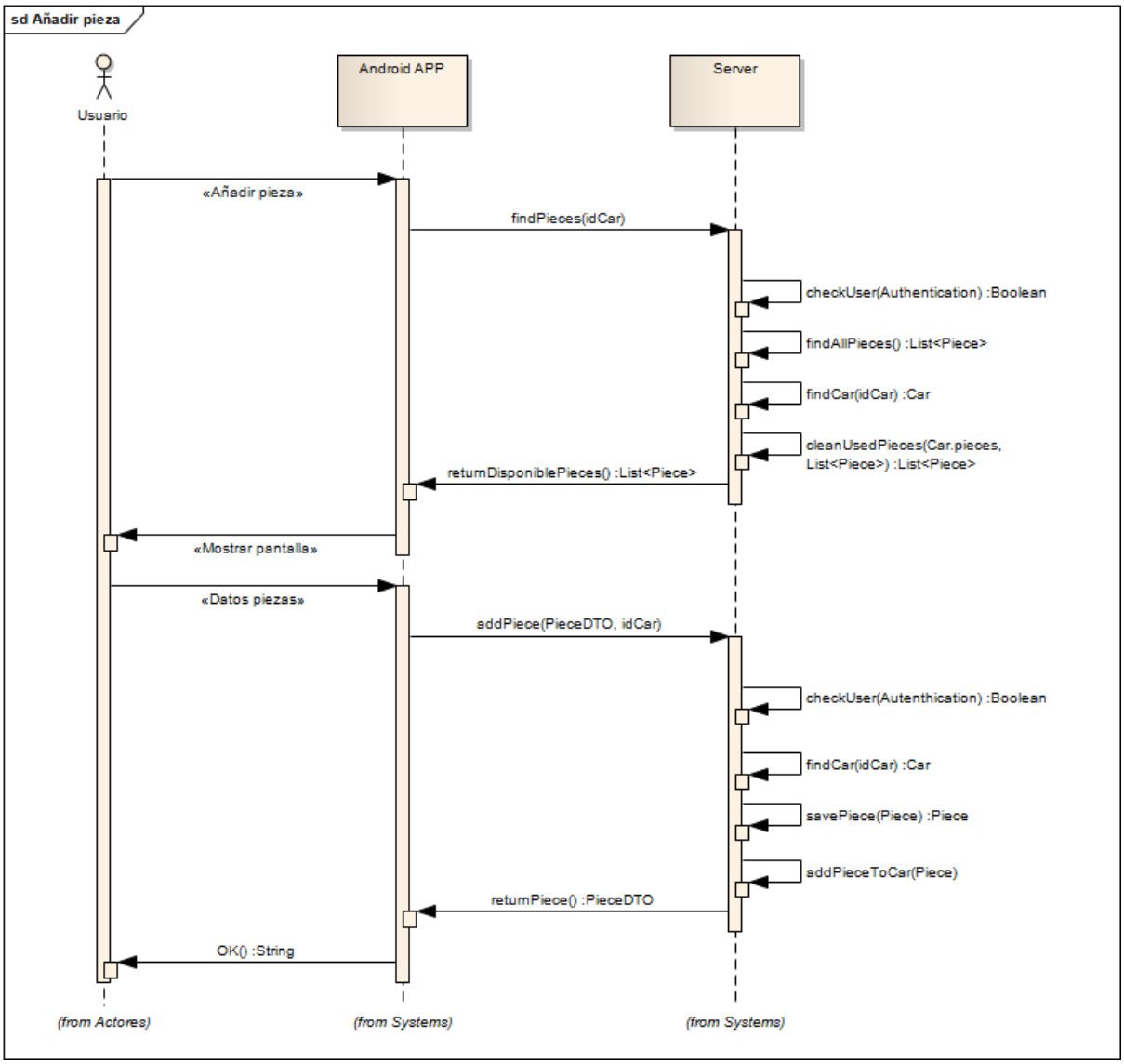

Figura 3.22: Diagrama de Interacción: Caso de Uso «Añadir Pieza»

3.2.3.4.2 CU Modificar Pieza

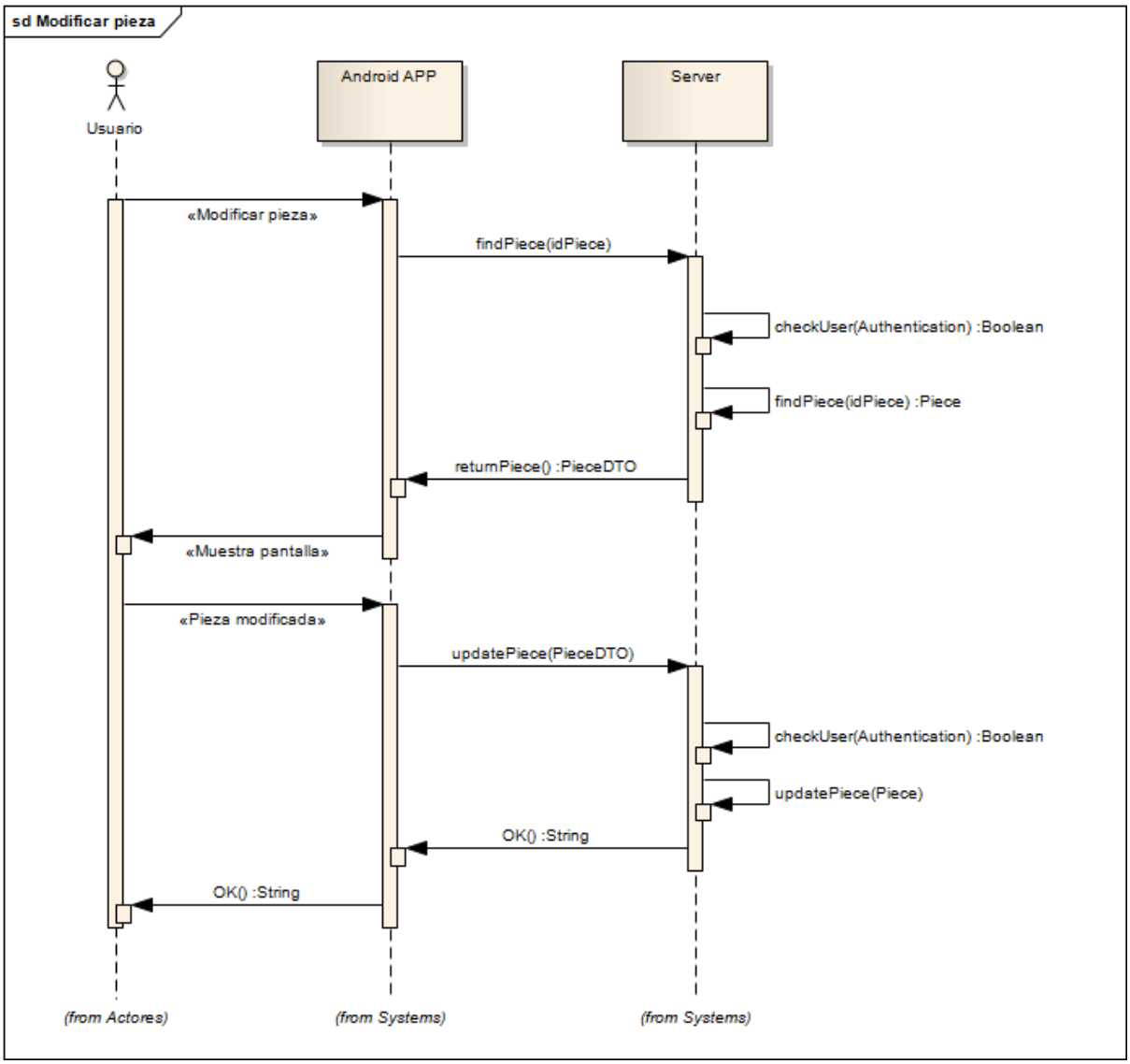

Figura 3.23: Diagrama de Interacción: Caso de Uso «Modificar Pieza»

3.2.3.4.3 CU Ver Pieza

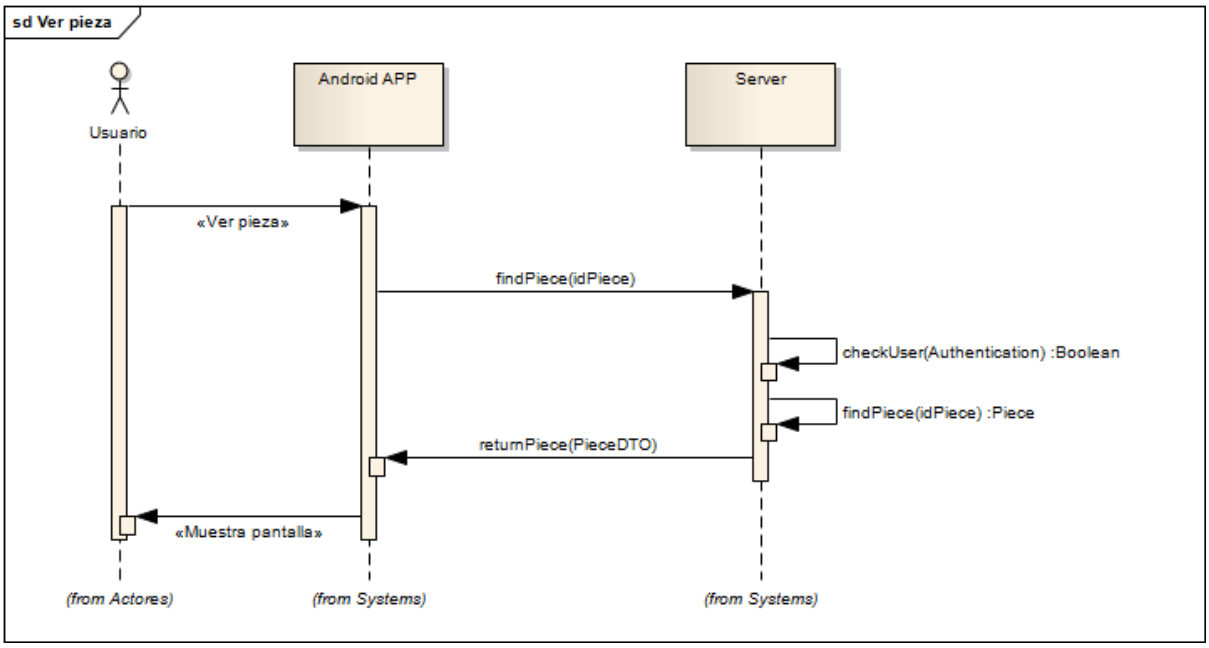

Figura 3.24: Diagrama de Interacción: Caso de Uso «Ver Pieza»

3.2.3.4.4 CU Eliminar Pieza

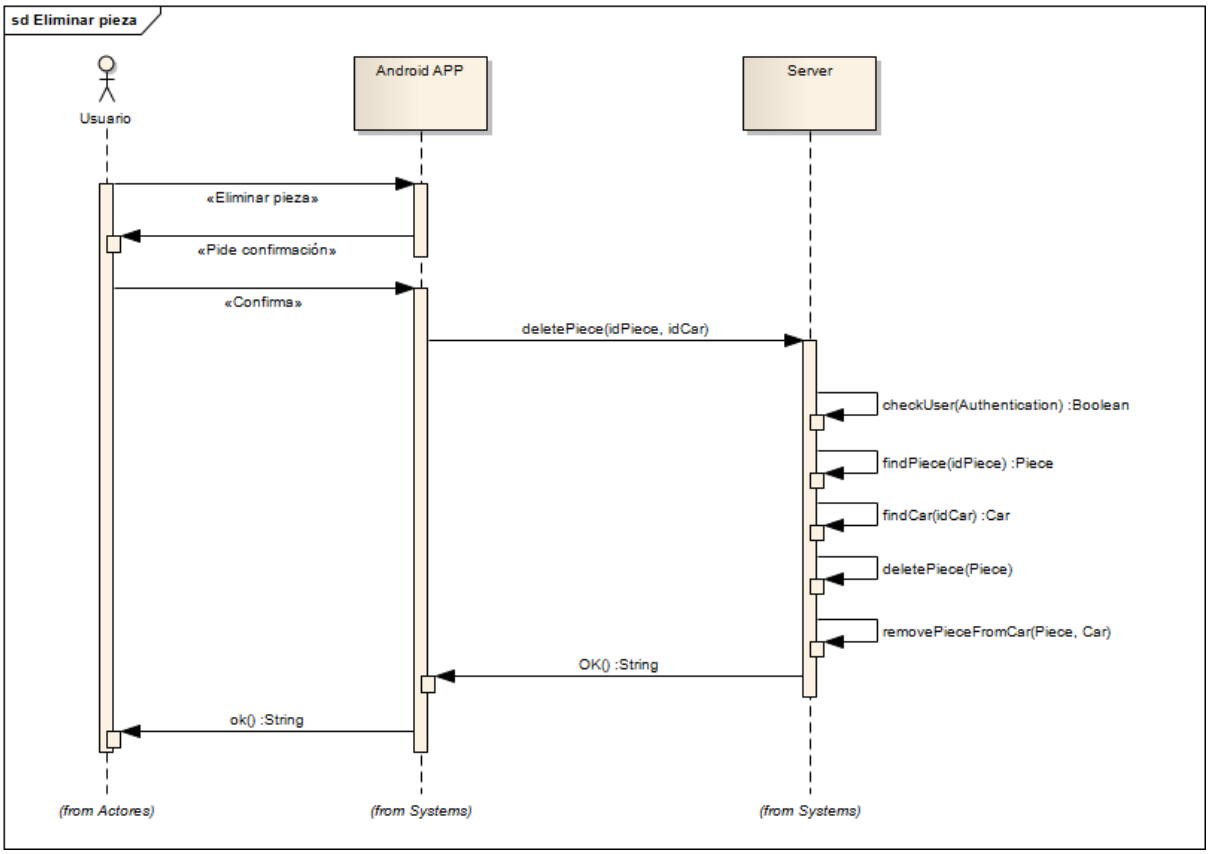

Figura 3.25: Diagrama de Interacción: Caso de Uso «Eliminar Pieza»

3.2.3.4.5 CU Consultar listado de piezas

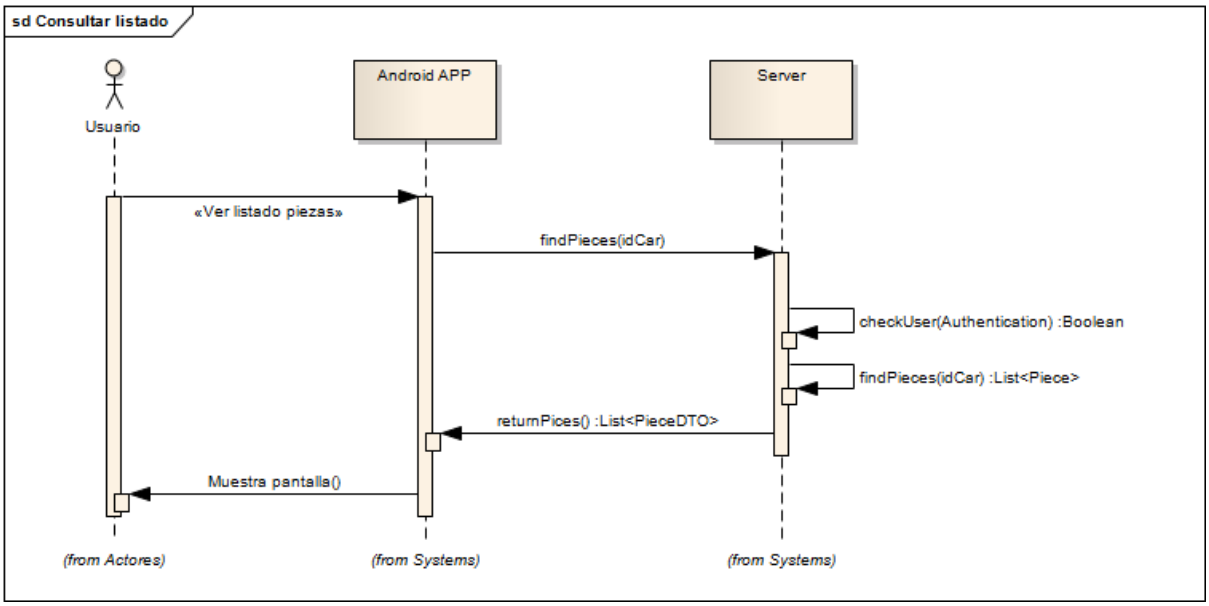

Figura 3.26: Diagrama de Interacción: Caso de Uso «Consultar listado de piezas»

3.2.3.4.6 CU Sustituir Pieza

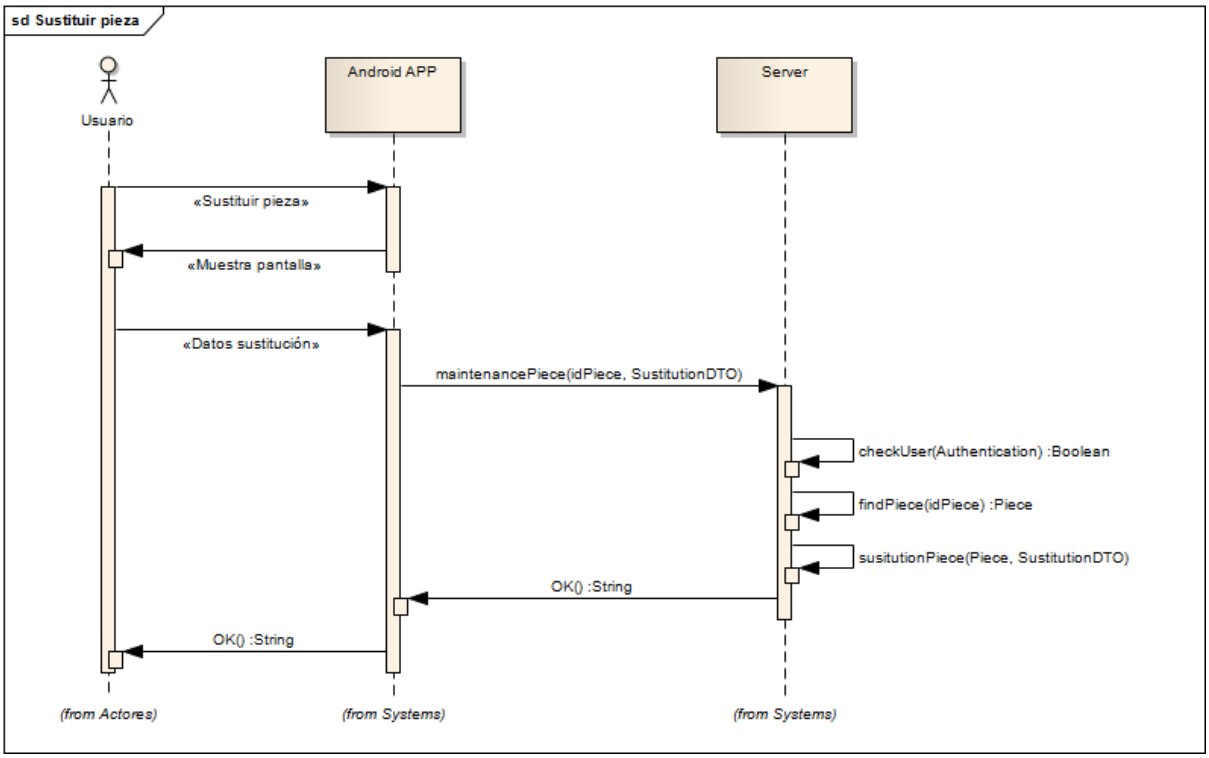

Figura 3.27: Diagrama de Interacción: Caso de Uso «Sustituir Pieza»

3.2.3.4.7 CU Buscar Pieza

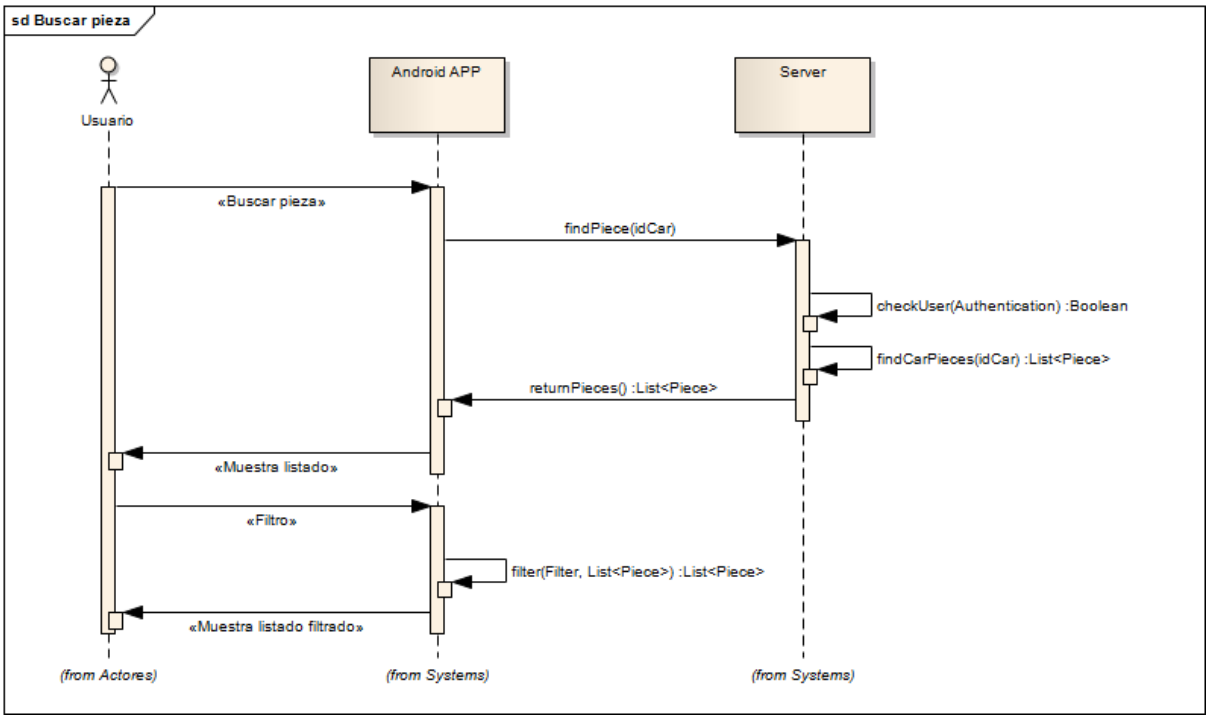

Figura 3.28: Diagrama de Interacción: Caso de Uso «Buscar Pieza»

3.2.3.5. Diagramas CU Consultar estadísticas

3.2.3.5.1 CU Ver estadísticas coche

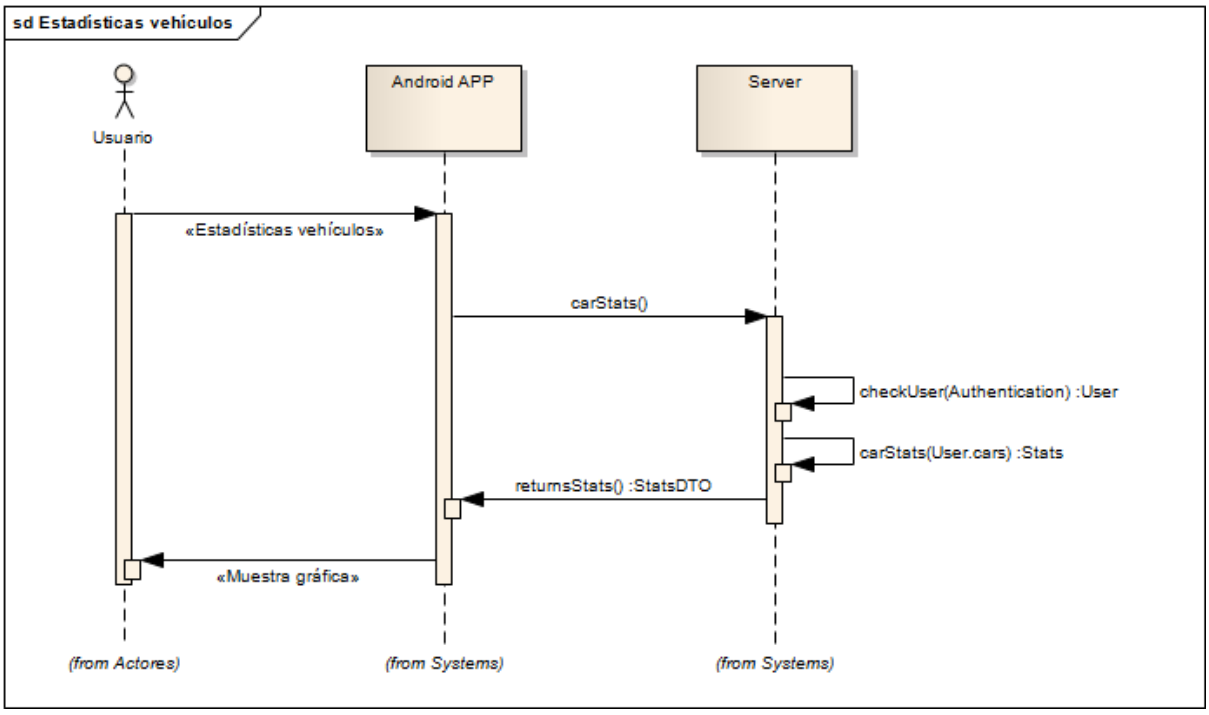

Figura 3.29: Diagrama de Interacción: Caso de Uso «Ver estadísticas coche»

3.2.3.5.2 CU Ver estadísticas combustible

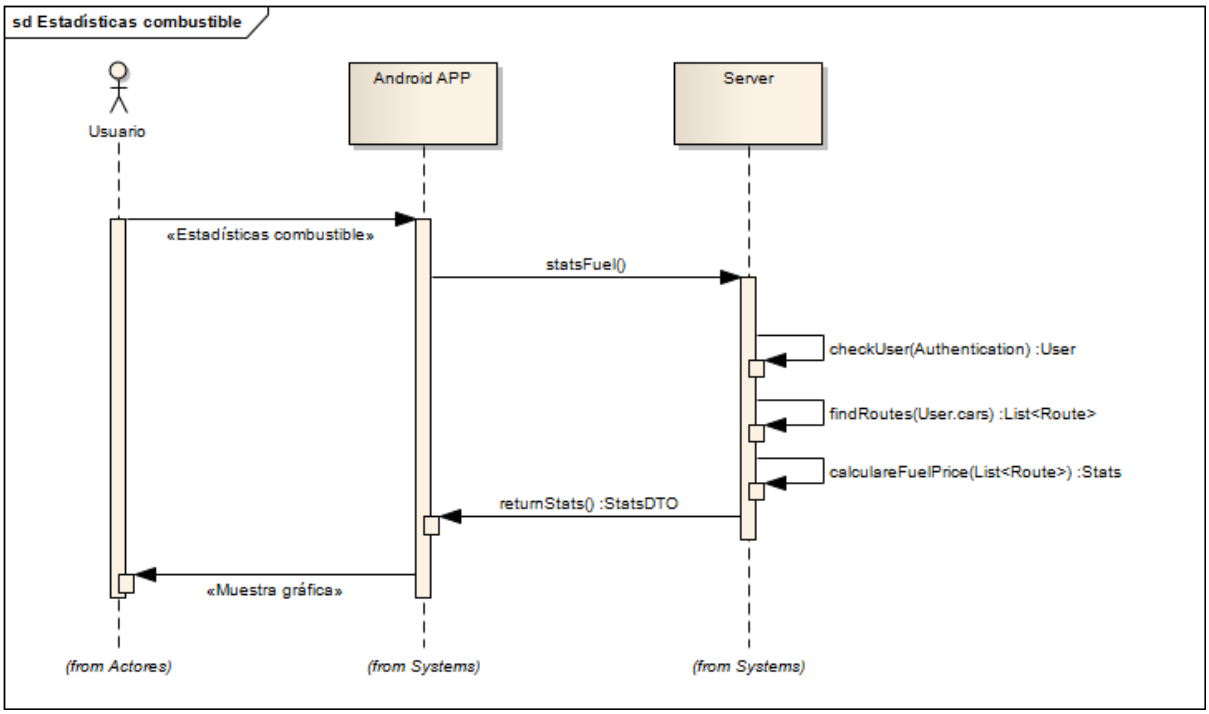

Figura 3.30: Diagrama de Interacción: Caso de Uso «Ver estadísticas combustible»

3.2.3.5.3 CU Ver listado de gráficas disponibles

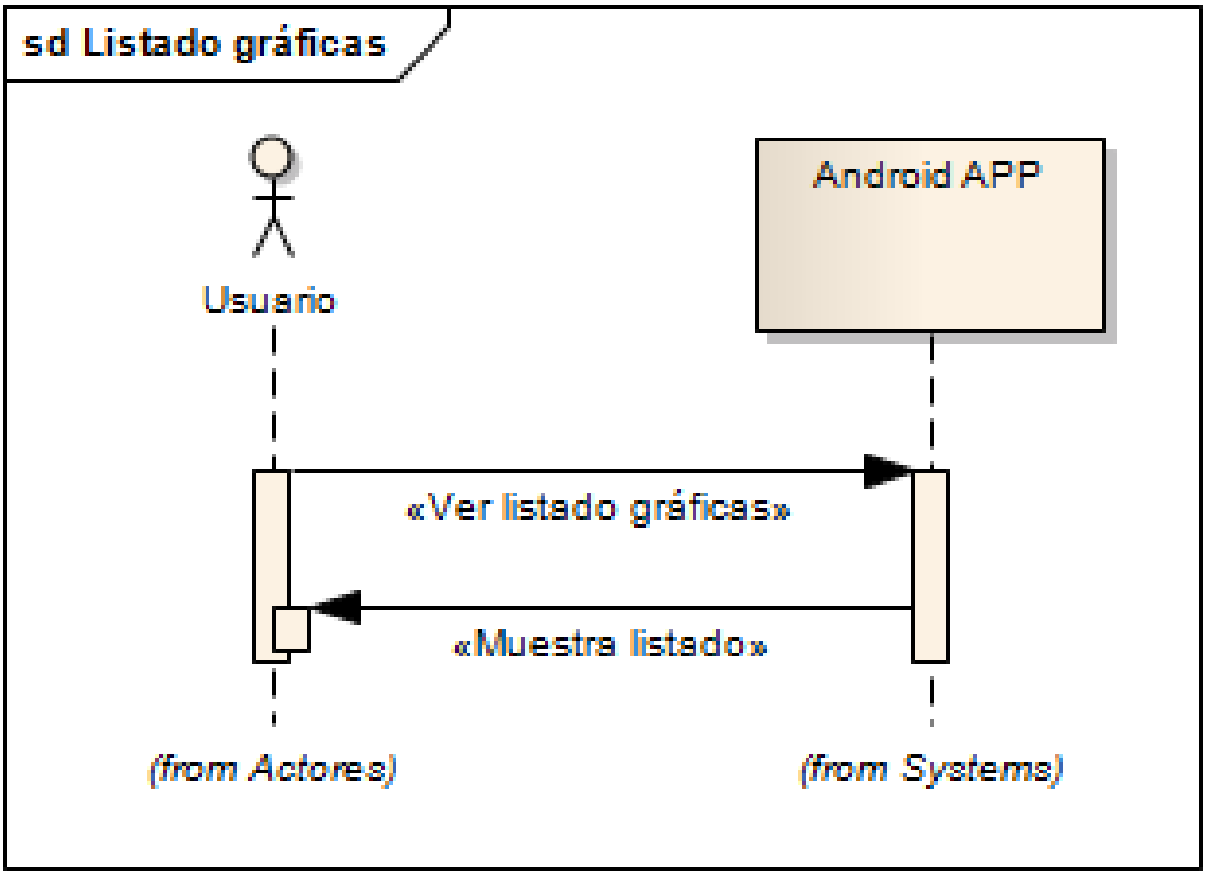

Figura 3.31: Diagrama de Interacción: Caso de Uso «Ver listado de gráficas disponibles»

3.2.3.5.4 CU Ver estadísticas distancias

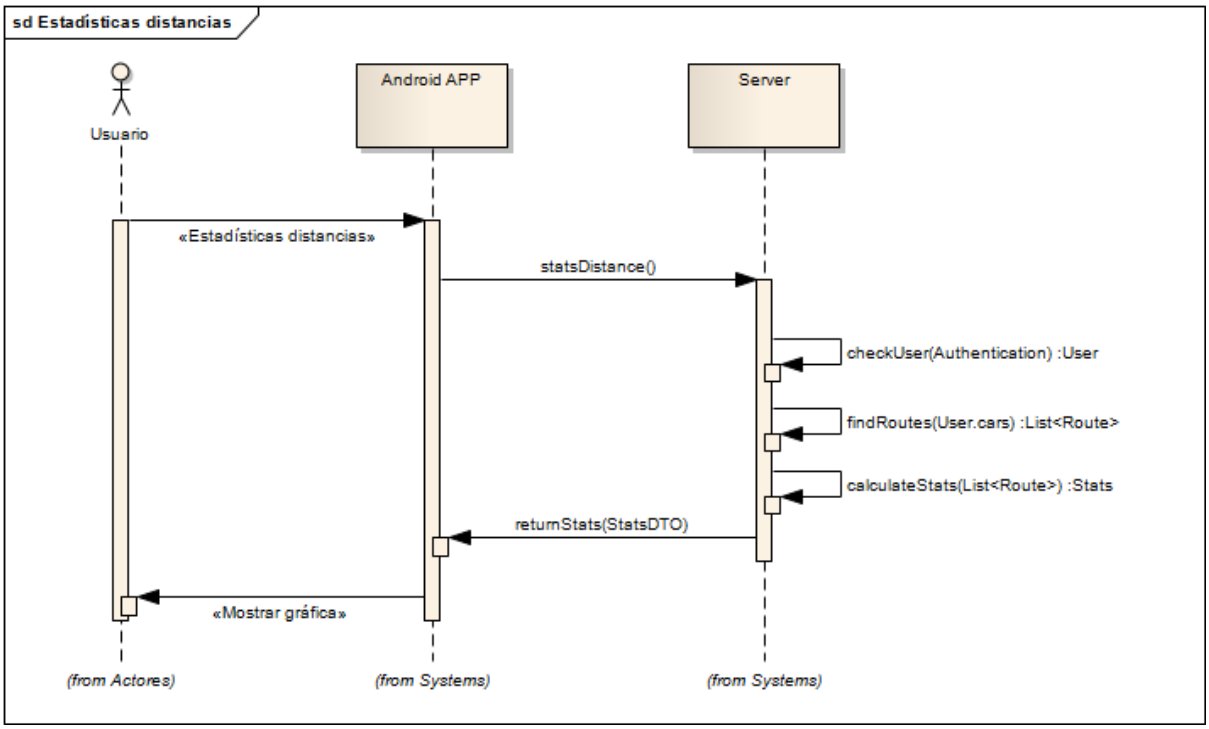

Figura 3.32: Diagrama de Interacción: Caso de Uso «Ver estadísticas distancias»

3.3. Requisitos de información

Los requisitos de información que nos encontramos son bastantes exigentes. Por un lado necesitamos almacenar toda la información individual y propia de cada usuario, cada vehículo, etc pero por otro lado es necesario guardar una base de datos general con los datos de todas las marcas y modelos de marcas existentes.

Para poder gestionar toda esta información de la forma más sencilla y eficiente posible se realiza una serie de relaciones entre clases detalladas en el siguiente esquema de clases UML.

Los datos a tener en cuenta para las distintas entidades son:

- Una serie de marcas.
- Una serie de modelos pertenecientes a una marca en concreto.
- Una serie de usuarios registrados en el sistema.
- Cada usuario podrá tener un número N de vehículos.
- Cada vehículo podrá haber realizado un número N de trayectos.
- Cada vehículo tendrá un número N de piezas.
- Cada pieza tendrá un número de N de cambios o sustituciones sufridas.

A continuación podemos ver el correspondiente diagrama de clases UML:

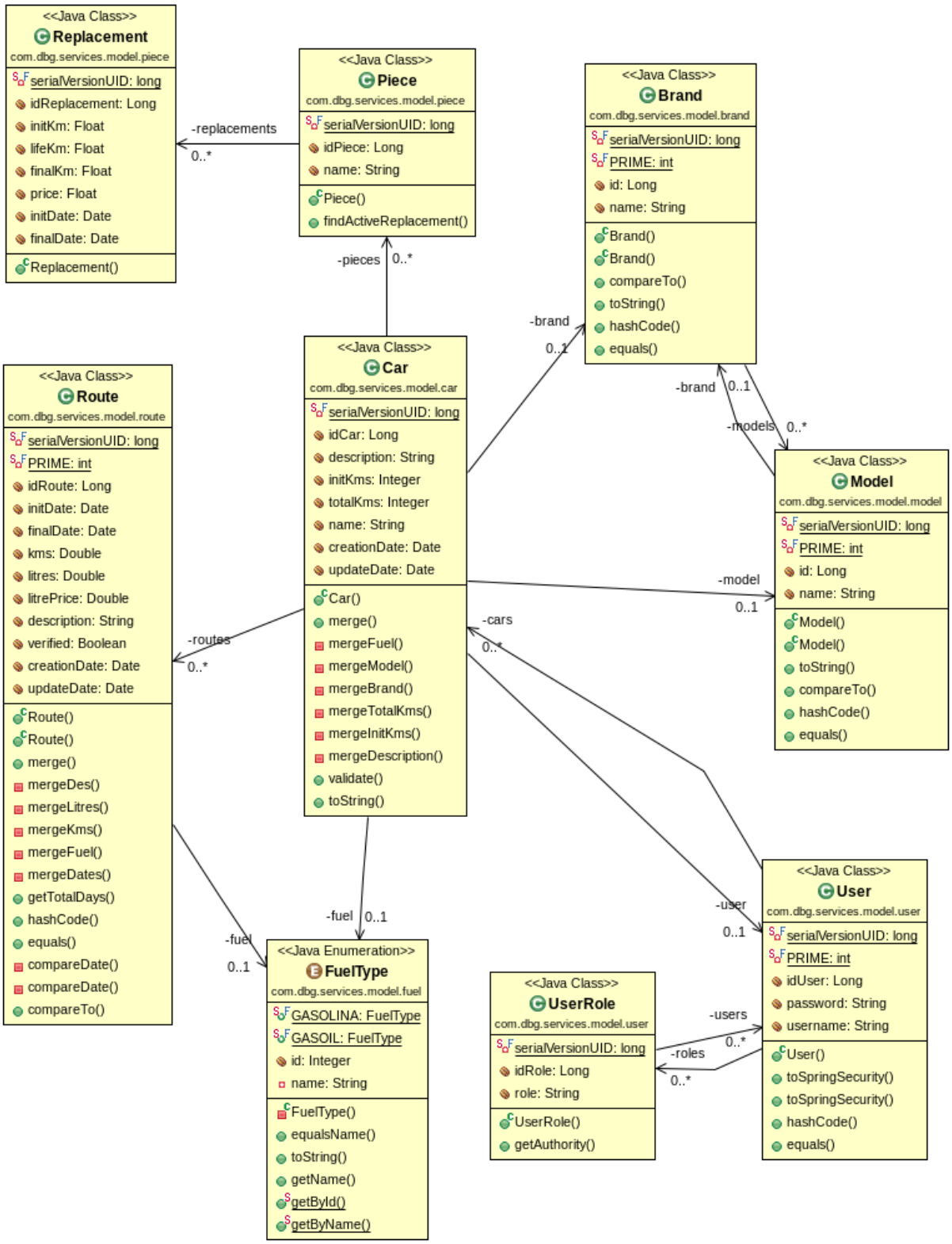

Figura 3.33: Diagrama conceptual de clases UML

3.4. Requisitos no funcionales

Los requisitos no funcionales son diversos, la mayoría buscan estabilidad, fiabilidad, alta disponibilidad y rendimiento.

- **Alta disponibilidad:** Aunque falle una instancia es importante no dejar de dar servicio.
- **Escalabilidad:** Posibilidad de escalar fácilmente el sistema añadiendo instancias a un cluster.
- **Seguridad:** Seguridad de la plataforma ante posibles ataques o pérdida de información.
- **Backup:** Copias de seguridad de datos.
- **Fácil comunicación:** Poder comunicar fácilmente desde diferentes plataformas para consultar datos.
- **Fiabilidad:** El sistema tiene que ser fiable, a parte de tener alta disponibilidad, debe ser lo más reducida posible la posibilidad de error.
- **Rendimiento:** El tiempo de respuesta tiene que ser lo más reducido posible. Lo ideal es estar siempre que sea posible por debajo del segundo.
- **Coste ajustado:** El coste de la infraestructura debe ser lo más ajustado posible. Utilización de código libre y en plataformas gratuitas.

Capítulo 4

Diseño del Sistema

4.1. Diseño de la arquitectura

A la hora de diseñar una aplicación nos encontramos con diversos factores. Necesitamos encontrar un punto de equilibrio donde se encuentre la mejor solución al problema. Para el diseño de esta aplicación existen varias premisas importantes que se deben cumplir estrictamente:

- Arquitectura física lo más barata posible.
- Aplicación móvil.
- Consistencia de datos.
- Adaptación al cambio.
- Facilidad a la hora de evolucionar la aplicación.

4.1.1. Arquitectura lógica

Para la elección de la arquitectura lógica se tienen en cuenta varias posibilidades. La elección final se decide valorando pros y contras.

Aplicación Android

La primera posibilidad que se trata es una aplicación nativa en Android con una Base de Datos SQL Lite donde almacenar la información. El problema de esta primera opción es la pérdida de datos.

Todas las aplicaciones con este tipo de arquitectura tienen un problema muy grande y es que la información sólo se encuentra en el dispositivo móvil. Esto hace que frente a la pérdida o cambio del dispositivo la información se perderá.

Otro problema de este tipo de sistemas es que no ofrecen gran posibilidad de cambio o adaptación futura. El tener todos los datos en la aplicación móvil hace imposible poder tener otros sistemas como aplicaciones web.

En este punto: se necesita una aplicación central donde almacenar toda la información. Con un sistema robusto, mantenible y con posibilidad al cambio.

Aplicación Servidor

Una vez clara la postura de crear una aplicación como servidor es hora de tener en cuenta los requisitos de la misma:

- **Comunicación multiplataforma:** Se necesitaba un sistema que facilitase la comunicación con la aplicación Android y en un futuro con otro tipo de sistemas.
- **Sistema de Base de Datos:** Se necesita un sistema barato, fácil de implantar pero a la vez robusto. El sistema elegido debe ser fácilmente modificado por otro más potente si fuese necesario.

Para la comunicación entre sistemas queda claro que se necesita algún tipo de servicios web. A la hora de elegir el tipo de Servicios Web decido entre Servicios con protocolo SOAP o REST con JSON.

Para trabajar con Android es recomendable usar JSON ya que es donde más documentación y facilidades podemos encontrar. También, valorando una futura versión web es recomendable utilizar JSON ya que cualquier Framework JS trabaja mejor. Otra de las ventajas del JSON es que los resultados de los servicios pesan mucho menos que las llamadas XML y cuanto más ligero sea el peso, generalmente, más barato será el coste de Red, procesamiento, etc.

Una vez ha quedado claro que se necesita un Servidor con llamadas REST que devuelve JSON. Una de las formas más limpias y claras de realizar un Servidor con llamadas REST siguiendo una estructura clara y limpia es un Servidor REST API. Este tipo de sistemas realiza llamadas POST, PUT, GET y DELETE, trabaja con servicios rest y se trata de un sistema comunmente utilizado junto a Android, por tanto, me decido por este tipo de sistema.

La base de datos, al principio, tendrá que ser una gratis y lo más potente posible, valoro PostgreSQL y MySQL. La decisión de utilizar MySQL es únicamente por simpatía.

Otra importante decisión es el lenguaje de programación a utilizar. Por diversas razones descarto cualquier tipo de lenguaje que no sea Open Source. Así que decido entre PHP, C++ y Java.

C++ se encuentra un poco anticuado en este aspecto, carece de Frameworks apliamente distribuidos y probados por lo que lo descarto.

PHP posee un buen Framework que aunque no es su función puede desempeñarla: Symfony 2. PHP nos ofrece la ventaja de la facilidad a la hora de realizar modificaciones, la seguridad nos la proporciona el propio Framework. El motivo de descartar PHP es la dimensión del proyecto, se trata de una aplicación bastante grande, con muchas variables que hace a Java una mejor opción por mayor disponibilidad de Frameworks, adaptación a diferentes sistemas, poco acoplamiento, etc.

Entre los Frameworks para Java necesito algo para la seguridad de los servicios REST, algún tipo de ORM para abstraer el tipo de Base de Datos y poder utilizar alguna más potente en el futuro. Puesto que realizar las llamadas a través de Servlets es algo muy tedioso, necesito algún Framework para realizar las llamadas.

Para la inicialización de las clases se utiliza Spring Core ya que facilita la inyección de dependencias de forma muy sencilla.

Como ORM es elegido Hibernate para la persistencia y Spring como manejador de las transacciones.

Para la seguridad se opta por Spring Security ya que provee todo lo necesario y es fácilmente integrable.

Para los servicios REST elijo Spring MVC ya que se integra muy fácilmente con Spring Core y Spring Security.

4.1.2. Arquitectura física

4.1.2.1. Infraestructura

Para la arquitectura física existen varias premisas:

- **Rapidez:** Es muy importante un tiempo de respuesta muy bajo ya que es la única forma de no hacer esperar al usuario.
- **Alta disponibilidad:** Antes posibles fallos sería importante un sistema con alta disponibilidad para evitar dejar de prestar servicio ante posibles fallos.
- **Escalabilidad:** Poder mejorar el rendimiento de forma fácil y rápida.

Para este punto queda claro que es necesario un Cluster con al menos dos servidores, de forma que si alguno deja de prestar servicio, el otro seguirá respondiendo a las peticiones. A la hora de escalar el sistema, los clusters son una decisión muy positiva ya que esta tarea se hace «en caliente» e inmediata.

La primera opción a evaluar es una solución en la nube, se tiene en cuenta AWS y el APP Engine de Google.

Los precios mensuales son para AWS de Amazon 75.70$ con un descuento los primeros 12 meses. Para el APP Engine de Google los precios son: 42.42$ al mes. Estos precios son para dos instancias para el Servidor, un balanceador y un MySQL, todos ellos en las máquinas virtuales más pequeñas de ambas empresas.

Los precios calculados pueden verse en sus correspondientes páginas web:

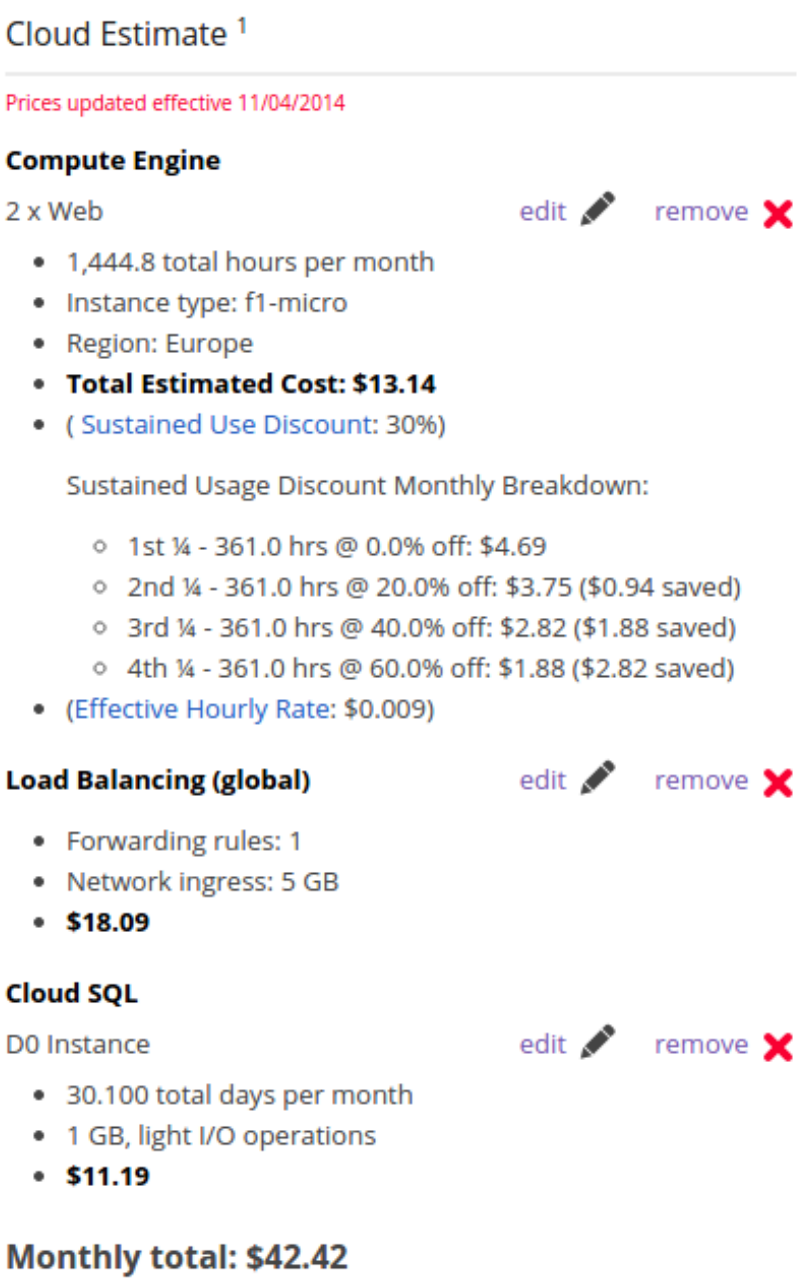

Figura 4.1: Estimación de coste mensual de la infraestructura en el APP Engine de Google.

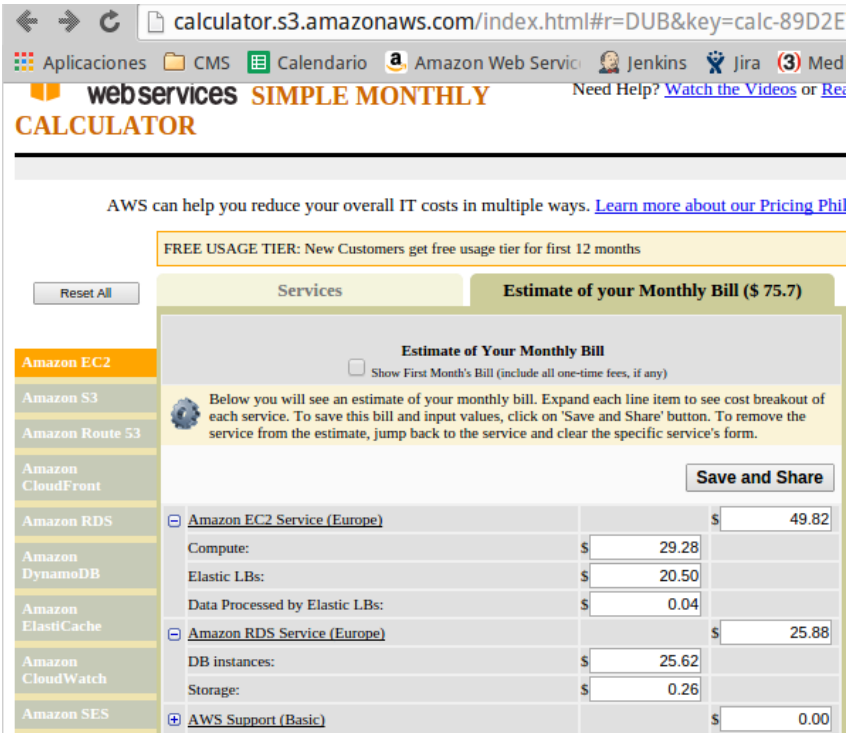

Figura 4.2: Estimación de coste mensual de la infraestructura en Amazon Web Services (AWS)

El más barato de los precios es 42.42$ al mes, lo que supone un coste de 509.04$. Debido al alto coste calculo otras posibilidades más baratas.

Las máquinas ofrecidas por Google son máquinas virtuales con 600Mb de RAM y procesador virtual compartido con otras instancias. Una máquina parecida podría ser una Raspberry Pi.

El precio de una Raspberry Pi es de unos 34$. Una vez la aplicación está terminada, tras realizar pruebas de carga con una instancia virtual con las mismas características y ver que todo funciona correctamente me decido por esta opción.

Roll over image to zoom in

Raspberry Pi Model B+ (B PLUS) 512MB Computer Board

by Raspberry Pi

228 customer reviews | 33 answered questions

Price: $34.38 & **FREE Shipping** on orders over $35. Details

In Stock.

Sold by Emerging Tech Sales and Fulfilled by Amazon. Gift-wrap available.

Want it Tuesday, Jan. 6? Order within **30 hrs 9 mins** and choose **One-Day Shipping** at checkout. Details

- 700MHz Broadcom BCM2835 CPU / 512 MB SDRAM @ 400 MHz / 10/100 Ethernet RJ45 On Board Network
- 40 pin Extended GPIO / Full Size HDMI / 4 USB Ports / Micro SD Slot
- More Energy Efficiency (Less Power Required) / Improved Power Management: Manage More Devices from Your Pi!
- Bigger and Better projects via an Expanded GPIO Header (40 pins vs. 26)
- Increased connectivity - 2 Extra USB ports (making a total of 4) and a new 4-pole connector replace the existing analogue and composite video port on the Model B

› See more product details

Figura 4.3: Precio de una Raspberry Pi en www.amazon.com

Para poder realizar un sistema como el deseado necesitaríamos una raspberry que hiciera de balanceador y dos que hicieran de nodos. También sería conveniente tener una para MySQL pero para ahorrar costes el propio balanceador puede hacer de servidor de base de datos.

La cesta de la compra con todo lo necesario sería:

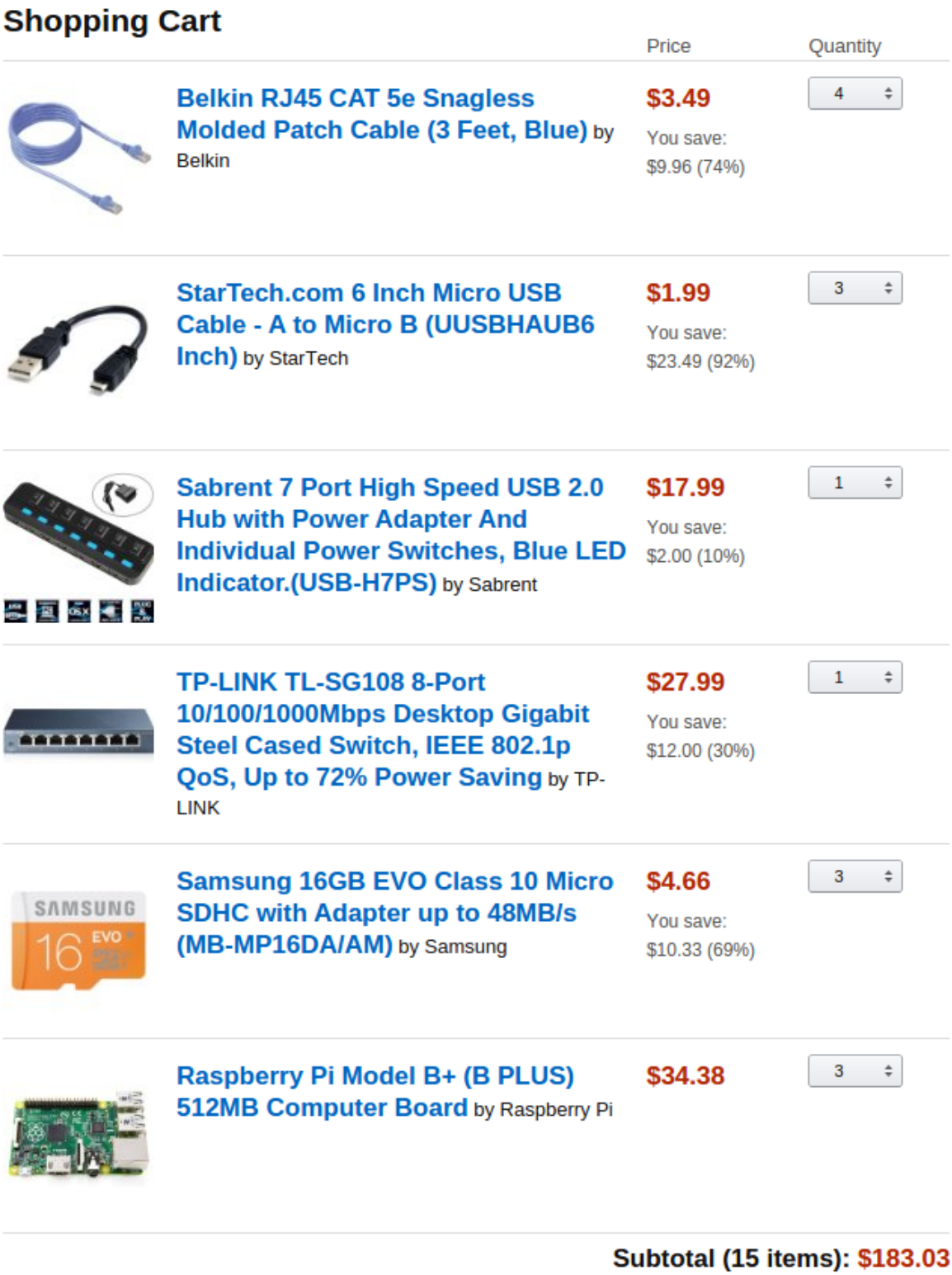

Figura 4.4: Compra de toda la infraestructura necesaria.

Además de esto, es necesario una conexión a internet con IP fija o algún sistema de dns dinámica. Puesto que se dispone de conexión domestica a internet y mi proveedor me proporciona IP fija, este punto estaría resuelto.

El coste total asciende a una sola compra de 183.03$, es decir, unos 150 euros.

El precio es muy inferior, por tanto, esta opción será la definitiva.

4.1.2.2. Configuración servidores

Para la instalación del sistema se dispone de tres Raspberry Pi, con sus respectivas Micro SD de 16GB. Un switch de ocho puertos con sus respectivos cables RJ45 para conectar a las Raspberrys. Para la alimentación eléctrica de las Raspberry opto por un hub alimentado de ocho puertos. Con conectar cada Raspberry Pi con un cable Micro USB sería suficiente.

Para la configuración:

Balanceador El balanceador será el encargado de recibir las peticiones. Para cumplir esta función será necesario dirigir todas las peticiones recibidas desde el exterior a esta Raspberry Pi. Esta función se realiza en la configuración del Router.

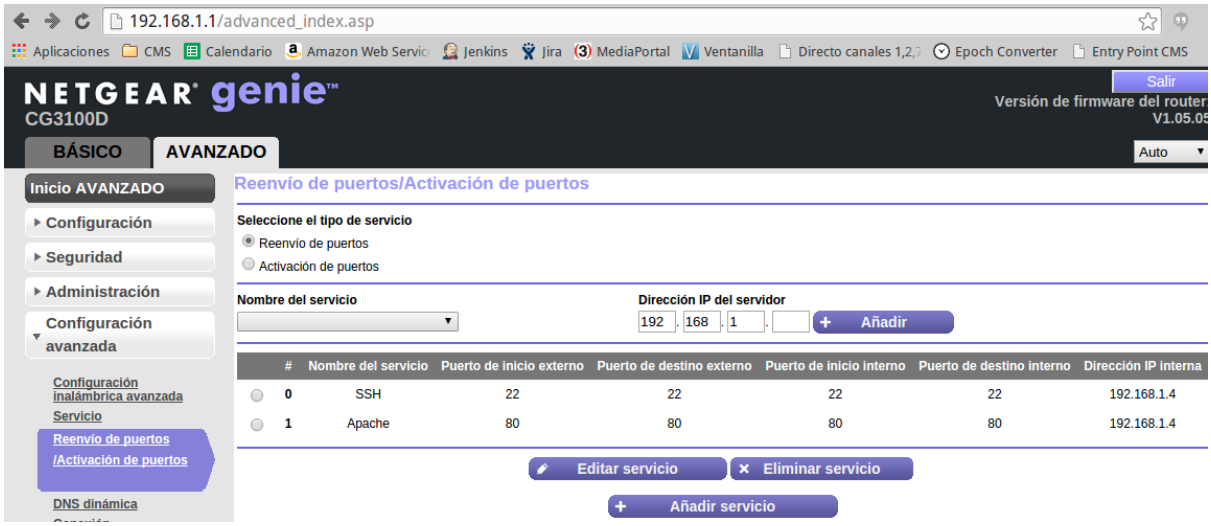

Figura 4.5: Configuración puestos del Router.

Se indica que todas las peticiones de conexión por SSH o HTTP sean recibidas por la máquina en la IP 192.168.1.4 que es el balanceador.

Para la configuración del mismo será necesario instalar MySQL 5.5., crear un usuario para las aplicaciones, es decir, un usuario con permisos para el esquema de la aplicación y habilitar las conexiones únicamente desde la red interna.

Como balanceador se instala un Apache con el módulo mod_jk. Este módulo es muy sencillo de configurar, sólo hay que indicar la IP de las máquinas que lo componen.

La configuración a continuación quedaría de la siguiente forma:

```
#
#------ ajp13_worker WORKER DEFINITION ------------------------------
#--------------------------------------------------------------------
#
  worker.list=router
  worker.router.type=lb
  worker.router.balance_workers=worker1,worker2
#
# Defining a worker named ajp13_worker and of type ajp13
# Note that the name and the type do not have to match.
#
  worker.worker1.type=ajp13
  worker.worker1.host=192.168.1.10
  worker.worker1.port=8009

  # Define the second member worker
  worker.worker2.type=ajp13
  worker.worker2.host=192.168.1.12
  worker.worker2.port=8009
```

Figura 4.6: Configuración puestos del Router.

Esto hace que escalar el sistema sea muy sencillo, sólo habría que comprar otra máquina e incluirla en la configuración del cluster.

Con todo esto ya estaría configurado el cluster.

Nodos Para la configuración de los nodos sólo se crea una configuración, se crea una imagen de la tarjeta Micro SD y se replica en todos los nodos deseados.

Para el funcionamiento de los nodos sólo será necesario realizar la instalación de la JDK de Java 1.7.

Posteriormente un Apache Tomcat 7 como servidor de aplicaciones. Puesto que tenemos un cluster será necesario configurar la réplica de sesión. La réplica de sesión es uno de los puntos más delicados e importantes ya que los usuario sólo deben loguearse en uno de los nodos. Podemos realizar la réplica de sesión por base de datos, memcaché o mediante broadcast, es decir, por la red interna.

Como opción óptima sería interesante tener un servidor de Memcached pero aumentaría el coste. Por tanto, mientras el sistema funcione de forma fluida opto por la opción de broadcast, es decir, réplica en red.

Una vez realizada esta opción ya sólo quedaría desplegar las aplicaciones en los diferentes nodos.

Para automatizar esta tarea lo máximo posible dispongo de una herramienta de integración continua, Jenkins.

Figura 4.7: Jenkins

Mediante un Script de despliegue Jenkins realiza la compilación del servidor gracias a Maven, posteriormente realiza los test y en caso de éxito los copia a los nodos y realiza el despliegue de los mismo.

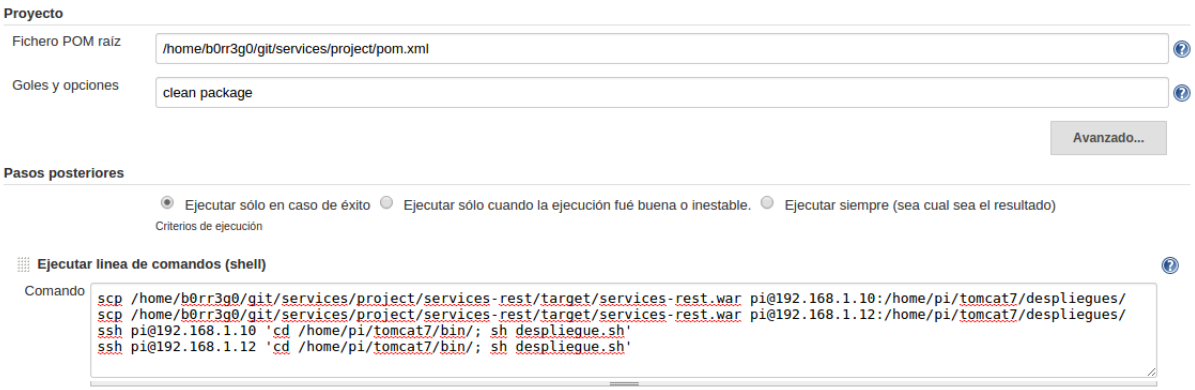

Figura 4.8: Configuración despliegues en Jenkins

El fichero despliegue.sh es un Script alojado en cada uno de los nodos, el Script se encarga de desplegar el fichero en tomcat. Para poder realizar todas estas operaciones sin necesidad de ser root o de introducir contraseña hay que añadir la clave pública de la máquina con Jenkins en el fichero /.ssh/authorized_keys de los nodos.

4.1.3. Arquitectura de diseño

Ambas aplicaciones han sido divididas en tres capas, una capa de integración, encargada de realizar todas las labores de recopilación de datos ya sea desde base de datos o desde otros sistemas. Una capa de negocio, servicios encargado de realizar toda la lógica de la aplicación y una capa de presentación.

El esquema general sería el siguiente:

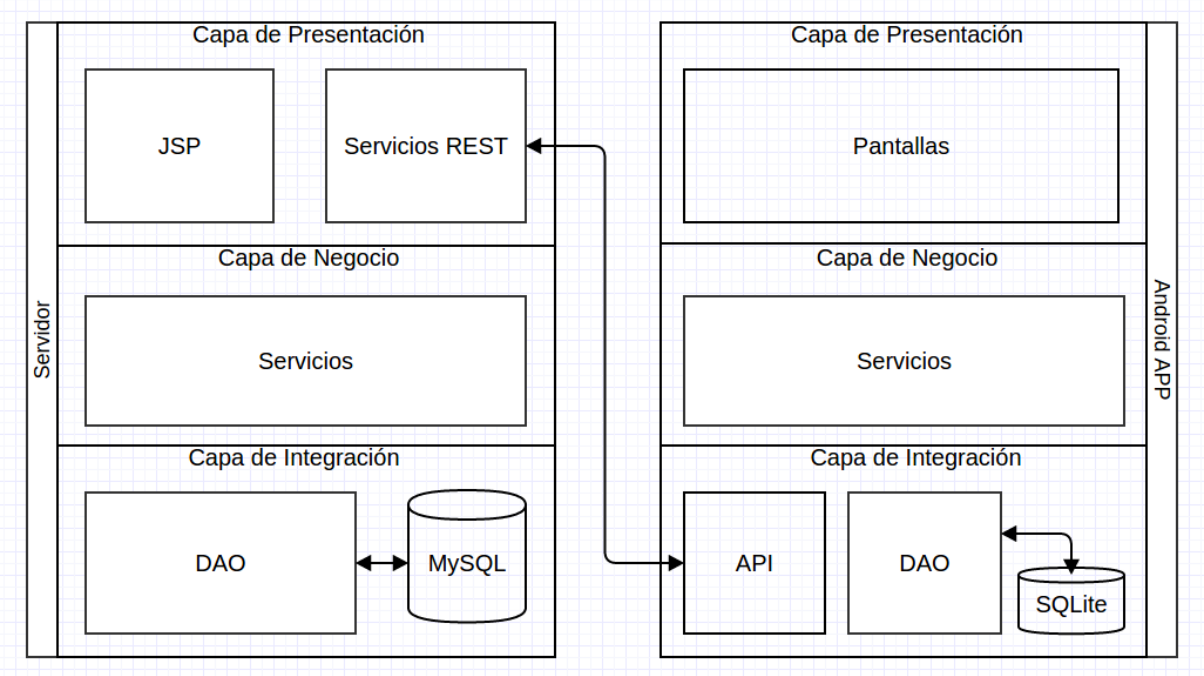

Figura 4.9: Configuración en capas del sistema.

4.1.3.1. Capa de Presentación

Servidor La capa de presentación en el servidor se puede dar de dos formas diferentes. Por un lado posee dos JSP, uno para el «login» y otro para el «logount». De esta forma se pueden realizar estas operaciones desde el navegador.

La otra posibilidad de la capa de presentación son todos los Servicios REST que exponen los datos.

APP Android La capa de presentación de la aplicación Android la componen todas y cada unas de las pantallas de la aplicación. Estás pantallas son detalladas en secciones posteriores.

4.1.3.2. Capa de Negocio

La capa de negocio tiene la misma función en ambos sistemas. Piden todos los datos necesarios, sea cual sea el sistema del que proceden. Una vez son tratados y están listos para ser mostrados, se mandan a la capa de presentación.

4.1.3.3. Capa de Integración

Servidor La capa de integración en el servidor se compone únicamente de todos los DAOS que realizan las peticiones a Base de Datos. Extraen toda la información necesaria de MySQL y devuelve los datos a la Capa de Negocio que es la única que puede realizar esta petición.

APP Android En el cliente la capa de integración la componen los DAO que acceden a SQLite y todas las llamadas que componen el API al servidor.

El DAO sólo accede a base de datos para recuperar datos de la sesión del usuario, como recordar el JSESSIONID o la contraseña, pero nunca almacena datos sensibles de perder. La contraseña es guardada encriptada por seguridad.

El API realiza todas las llamadas para recuperar los datos al servidor. La ventaja de tener un API es que todas las llamadas se encuentran centralizadas en un mismo lugar.

Una vez los datos son recuperados, se pasa el control a la capa de negocio.

4.2. Diseño de datos

4.2.1. Diagrama E/R Base de Datos

Servidor La estructura de datos compleja se encuentra en el Servidor, aquí es donde se encuentran todos los datos.

Para el almacenamiento se ha utilizado una base de datos MySQL. El diseño de la base de datos es el que se muestra a continuación:

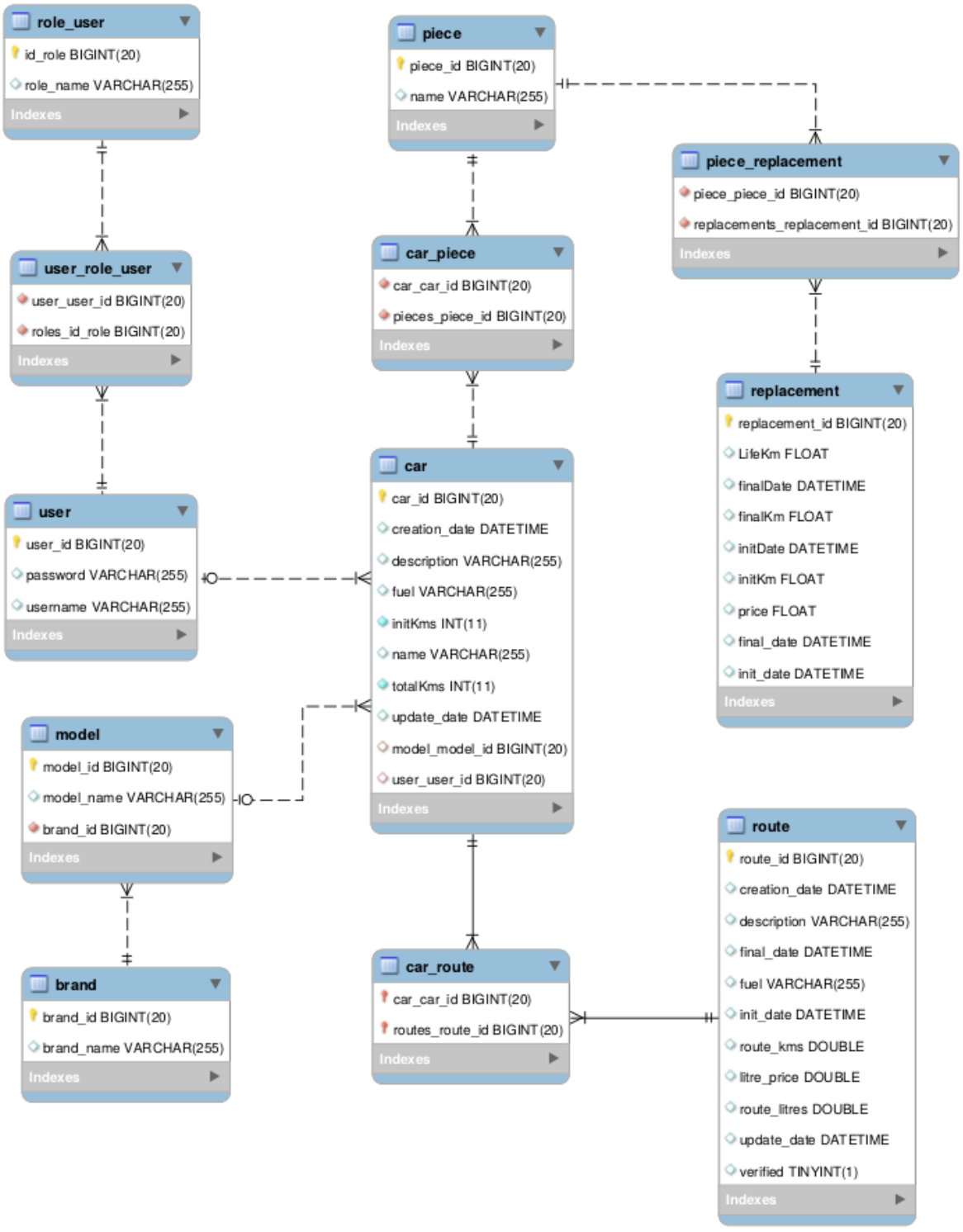

Figura 4.10: Diseño de la base de datos

Cliente Para el almacenamiento en cliente sólo se ha utilizado una tabla de apoyo en SQLite.

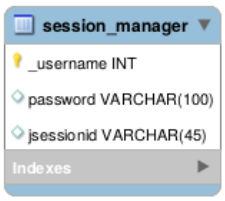

Figura 4.11: Diseño de la base de datos en SQLite para Android

4.2.2. Diseño lógico de la Base de Datos

Para el cliente Android, al sólo tener una tabla su definición es trivial, por tanto me centraré en la definición de la base de datos MySQL del servidor.

A continuación podemos ver la definición de todas las tablas:

Roles Para controlar los roles existentes en la aplicación existe la tabla «user_role». Aquí se dan de alta los roles que los usuarios podrán tener en la aplicación:

Role_user				
Atributo	Tipo	PK	FK	Nulo
id_role	BIGINT(20)	Si	No	No
role_name	VARCHAR(255)	No	No	No

Cuadro 4.1: Tabla Role_user

Usuarios

User				
Atributo	Tipo	PK	FK	Nulo
user_id	BIGINT(20)	Si	No	No
password	VARCHAR(255)	No	No	No
username	VARCHAR(255)	No	No	No

Cuadro 4.2: Tabla User

La tabla usuarios posee una relación uno a muchos con la tabla «Role_user», esta relación es la encargada de controlar qué roles tiene un usuario en el sistema. Dicha relación se hace a través de la tabla «user_role_user», la cuál podemos ver a continuación.

User_role_user				
Atributo	Tipo	PK	FK	Nulo
user_user_id	BIGINT(20)	Si	Si	No
roles_id_role	BIGINT(20)	Si	Si	No

Cuadro 4.3: Tabla User_role_user

Modelos

Todas las marcas de vehículos tienen más de un modelo, en esta tabla se guardan los modelos.

Model				
Atributo	Tipo	PK	FK	Nulo
model_id	BIGINT(20)	Si	No	No
model_name	VARCHAR(255)	No	No	No
brand_id	BIGINT(20)	No	Si	No

Cuadro 4.4: Tabla Model

Esta tabla tiene una relación con la tabla «brand» de modo que cada modelo pertenece a una marca.

Marcas

Cada vehículo es de una marca concreta, todas las marcas son almacenadas en la siguiente tabla:

Brand				
Atributo	Tipo	PK	FK	Nulo
brand_id	BIGINT(20)	Si	No	No
brand_name	VARCHAR(255)	No	No	No

Cuadro 4.5: Tabla Brand

Debido a la relación que poseen los modelos ya podemos sacar todos los modelos de una marca.

Rutas

Todas y cada una de las rutas que realizan los vehículos son almacenadas en esta tabla. Esta es la definición de la misma:

Route				
Atributo	Tipo	PK	FK	Nulo
route_id	BIGINT(20)	Si	No	No
description	VARCHAR(255)	No	No	Si
final_date	DATETIME	No	No	Si
fuel	VARCHAR(255)	No	No	Si
init_date	DATETIME	No	No	Si
route_kms	DOUBLE	No	No	Si
litre_price	DOUBLE	No	No	Si
route_litres	DOUBLE	No	No	Si
creation_date	DATETIME	No	No	No
update_date	DATETIME	No	No	Si
verified	TINYINT(1)	No	No	Si

Cuadro 4.6: Tabla Route

Todos los trayectos pertenecen a un vehículo, explicaré esta relación en la tabla «car».

Piezas

Todos los vehículos poseen más de una pieza, esta tabla es la siguiente:

Piece				
Atributo	Tipo	PK	FK	Nulo
piece_id	BIGINT(20)	Si	No	No
name	VARCHAR(255)	No	No	No

Cuadro 4.7: Tabla Piece

Todas las piezas tienen una duración determinada, es decir, el aceite hay que cambiarlo cada cierto tiempo, etc. Por tanto esto sugiere que cada pieza tendrá un historial, una serie de cambios, estos cambios se almacenan en la tabla «replacement»:

Replacement				
Atributo	Tipo	PK	FK	Nulo
replacement_id	BIGINT(20)	Si	No	No
life_km	FLOAT	No	No	No
final_date	DATETIME	No	No	No
init_date	DATETIME	No	No	No
init_km	FLOAT	No	No	No
final_km	FLOAT	No	No	No
price	FLOAT	No	No	No

Cuadro 4.8: Tabla Replacement

Puesto que cada pieza tiene al menos un «replacement» pero puede tener n, existen una realacion uno a muchos. Para esta relación existe una tabla intermedia, la tabla «piece_replacement»

Piece_replacement				
Atributo	Tipo	PK	FK	Nulo
piece_piece_id	BIGINT(20)	Si	Si	No
replacements_replacement_id	BIGINT(20)	Si	Si	No

Cuadro 4.9: Tabla Piece_replacement

Vehículos

Los vehículos conforman la parte principal, todas las tablas de una forma u otra prestan algún servicio a la entidad vehículo. La tabla de vehículos, con nombre «car» es la siguiente:

Car				
Atributo	Tipo	PK	FK	Nulo
car_id	BIGINT(20)	Si	No	No
description	VARCHAR(255)	No	No	Si
fuel	VARCHAR(255)	No	No	Si
init_kms	INT(11)	No	No	No
name	VARCHAR(255)	No	No	Si
total_kms	INT(11)	No	No	No
model_model_id	BIGINT(20)	No	Si	No
user_user_id	BIGINT(20)	No	Si	No
update_date	DATETIME	No	No	No
creation_date	DATETIME	No	No	No

Cuadro 4.10: Tabla Car

Cada vehículo pertenece a un usuario determinado, por eso la relación uno a uno entre la tabla «car» y «user». Gracias a esta relación podríamos conocer los roles del usuario.

Un vehículo es de una determinada marca, que a su vez tiene un determinado modelo. Como dado el modelo se conoce su marca y queremos un diseño en 3FN la tabla «car» tiene una relación uno a uno con la tabla «model» de esta forma podemos conocer su modelo y a través de su modelo, su nombre.

Cada vehículo tiene una relación de trayectos. La tabla «route» es la encargada de almacenar los trayectos. Puesto que un vehículo puede tener más de un trayecto, existe una relación uno a muchos, por eso empleo una tabla para realizar esta relación. La tabla utilizada es «car_route»:

Car_route				
Atributo	Tipo	PK	FK	Nulo
car_car_id	BIGINT(20)	Si	Si	No
routes_route_id	BIGINT(20)	Si	Si	No

Cuadro 4.11: Tabla Car_route

Por otro lado, cada vehículo posee varias piezas, eso supone una relación uno a muchos. Del mismo modo, esta relación se realiza con ayuda de una tabla intermedia llamada «car_piece»:

Car_piece				
Atributo	Tipo	PK	FK	Nulo
car_car_id	BIGINT(20)	Si	Si	No
pieces_piece_id	BIGINT(20)	Si	Si	No

Cuadro 4.12: Tabla Car_piece

De esta forma quedaría realizado todo el diseño de la Base de datos.

4.3. Diseño de componentes

El diseño de los componenetes del sistema se ha divido en dos sistemas: el servidor y el cliente.

Para ambos diseño se ha procedido a una descomposición en capas, siguiendo el patrón MVC.

Un aspecto a tener en cuenta es que el diseño es independiente del lenguaje de programación utilizado. En todo el proceso de diseño se pueden ver aspectos orientados a técnicas específicas de Java para poder facilitar la implementación y claridad del código. El ejemplo más claro son las interfaces. En caso de querer utilizar otro lenguaje sería muy sencillo ya que las interfaces sólo son definiciones de método a implementar fácilmente sustituibles por una clase a heredar en otros lenguajes.

4.3.1. Servidor

Como ya se ha indicado en apartados anteriores, el proyecto del servidor se ha dividido en tres módulos. Un módulo para las clases del modelo, los POJOs. Otro módulo para los DTOs y un módulo con toda la lógica de negocio de la aplicación.

Los dos primeros módulos se encuentran estructurado en paquetes según la entidad pero son módulos muy sencillos para centrar toda la lógica en el módulo con todo el negocio del proyecto. Éste módulo es el que detallo a continuación.

Para la capa del modelo se ha utilizado el patrón Factory para los accesos a base de datos. A través de Hibernate como ORM y Spring como manejador de transacciones creo una clase abstracta que implementa todas las operaciones básicas a realizar con las diferentes entidades del modelo.

De esta forma para realizar cualquier operacion sólo será necesario heredar de este DAO genérico.

En el siguiente diagrama se muestra como quedaría la capa de DAOs.

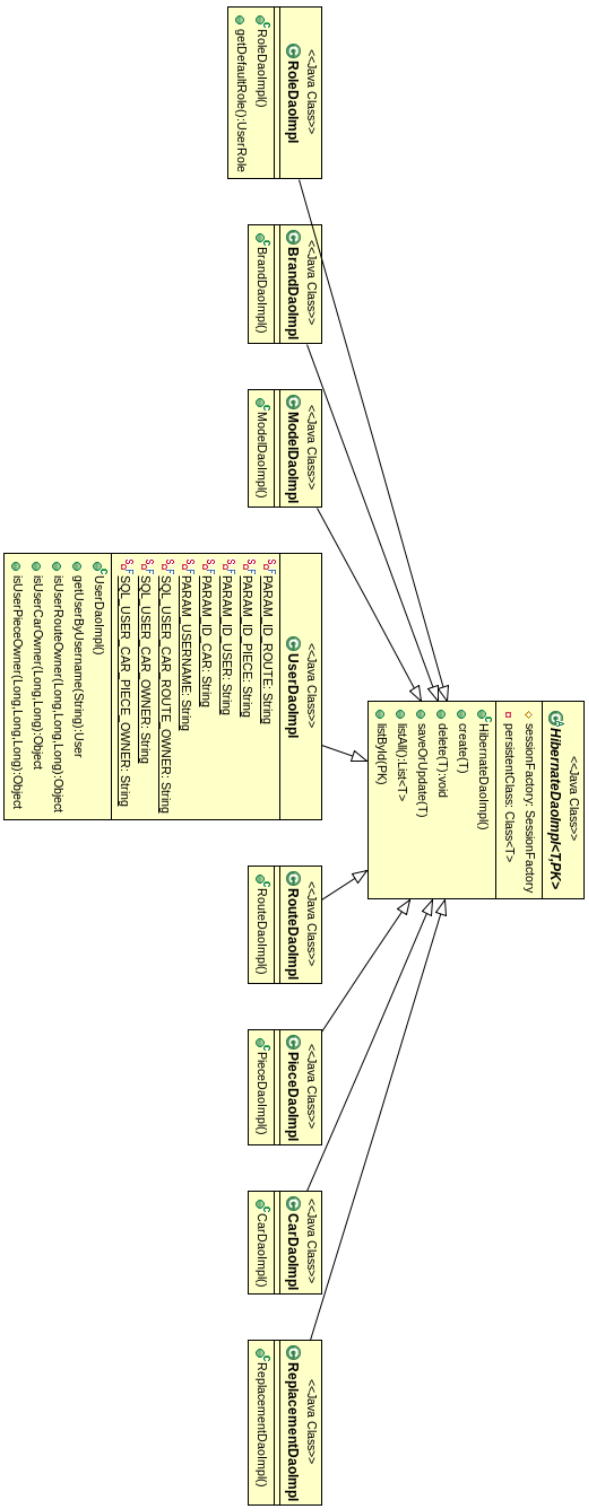

Figura 4.12: Diseño de la Capa de Integración en el Servidor

Otro de los aspectos destacables en esta capa es la creación de interfaces (en Java) para validar objetos o para actualizar. De esta forma la definición de esta función se encuentra centralizada, por ejemplo: todas las clases que se vayan a crear debería ser válida, de este modo implementan la interfaz «Validator» y consecuentemente implementan su método. En caso de utilizar otro lenguaje sustituiremos estas interfaces por otro

mecanismo similar.

Para la capa de servicios se utiliza el patrón Singleton ya que es el patrón por defecto de Spring Core. De esta forma sólo existe una instancia de cada servicio.

En todo el diseño es el servicio el que maneja la transacción con la base de datos. Esta parte ha sido realizada así para poder abrir y cerrar transacciones según la complejidad del servicio ya que las operaciones de los DAOS son siempre triviales.

Todos los servicios implementan las funciones de selección, actualización, creación y eliminación de datos que ofrecen los DAOs, de forma que es necesario pasar por el servicio para realizar estas operaciones. A parte de éstas, implementan servicios necesarios a lo largo de toda la aplicación.

Todas las funciones implementadas por los servicios se encuentran definidas previamente en las interfaces propias de dicho servicio. Por otra parte, existen varias interfaces genéricas para funciones que se realizan en varias entidades. Estas interfaces son por ejemplo «MapperService» o «ServiceInterface».

La interfaz «MapperService» es utilizada para todas las entidades del modelo que posean DTO, de esta forma proporciona la forma de la implementación del servicio de mapeo y en caso de ser necesario sólo habrá que implementar ésta interfaz.

La otra interfaz, «ServiceInterface» es la genérica para las entidades que posean DAO, es decir, las operaciones implementadas en el DAO son accesibles desde el servicio, pues todas estas operaciones se implementan gracias a ésta interfaz.

A parte de las interfaces genéricas existen las interfaces propias de cada uno de los servicios, por ejemplo la interfaz «PieceService» define otras operaciones propias de este servicio.

De esta forma quedaría definido el servicio independientemente de la implementación realizada. Si vemos el diseño de esta capa vemos que todo son servicios independientes de unos de otros.

La capa de la vista, es decir, todos los servicios REST son definidos y servidos gracias a Spring MVC. Todos estos «Controller» acceden directamente a servicios inicializados gracias a la inyección de dependencias de Spring.

Para estructurar todos los servicios de forma lógica y jerarquizada se ha utilizado la arquitectura REST cumpliendo estrictamente los siguientes niveles:

- **Uso correcto de URIs:** Las URLs deben estar correctamente jerarquizadas, por ejemplo, para recuperar una pieza se produce una jerarquía ya que una pieza pertenece a un vehículo y un vehículo a un usuario, de esta forma la url para recuperar una pieza quedaría de la siguiente forma: /services/user/car/idCar/piece/idPiece
- **Uso correcto de HTTP Verbs:** Aquí surge uno de los fallos más comunes en el uso de este tipo de sistemas. Cada una de las operaciones básicas debe ser realizada a través del verbo correspondiente tal y como sigue el estándar:

- **GET:** Utilizado para recuperar cualquier dato deseado. Recibe en la url el id del objeto a recibir, por ejemplo: /services/user/car/idCar/piece/idPiece es una forma de recuperar la pieza «idPiece» del vehículo «idCar».
- **POST:** A través de este verbo se deben realizar todas las operaciones de creación. Recibe el objeto a crear en el «body». Un ejemplo de uso sería: /services/user/car/idCar/piece/ recibiendo el vehículo al que se le añade la pieza y la pieza a crear en el «body».
- **PUT:** Verbo utilizado para la actualización de elementos. Su url sería: /services/user/car/idCar/piece/idPiece, es decir, similar a la de GET pero recibiendo en el body el objeto a actualizar.
- **DELETE:** Verbo utilizado para la eliminación de objetos. Su url es /services/user/car/idCar/piece/idPiece y eliminaría la pieza cuyo id fuese «idPiece».
- **PATH:** Para editar partes concretas de un recurso. Este método no ha sido implementado en el sistema actual ya que no es necesario para un sistema cuyas entidades son pequeñas.

Como parte de la seguridad y eficiencia de la aplicación los controller son el punto más delicado ya que supone el punto de entrada de cualquier petición o posible ataque. Por este motivo es muy importante realizar el mayor número posible de comprobaciones para nunca devolver datos a una persona no logueado o datos no pertenecientes a esta persona.

Un ejemplo de comprobaciones a realizar: cuando recibimos una petición hay que comprobar qué usuario la hace, si está pidiendo datos de él, si todos los datos a los que accede lo hace de forma lógica, es decir, de forma que permita la aplicación. En caso de un acceso indebido, sería importante controlarlo y poder implementar mecanismos para evitar que se repita, por ejemplo una «black list».

Una de las ventajas de utilizar Spring Security con Spring MVC es este punto. A partir de una serie de roles se puede indicar a que servicios o URLs puede acceder a un usuario de forma anónima o con uno u otro rol.

Para este sistema se encuentran todas las URL que manejan datos con el role «ROLE_USER» y las URLs de comprobación de login, registro o login con acceso abierto para cuanquier usuario.

Otro aspecto a tener en cuenta es implementar sistemas de caché en aquellos puntos más delicados ya que antes ataques de denegación de servicio o horas puntas de peticiones suponen un punto delicado en la plataforma.

4.3.2. Cliente

Para el diseño del cliente se han seguido las reglas del SDK de Android y un patrón de diseño MVC, concretamente MVP.

La diferencia esencial entre MVC y MVP es que MVP no es un patrón de diseño en si, sino una variante o modificación del MVC en la que la vista es totalmente independiente del modelo, siendo el presentador el que interactúa entre modelo y vista.

Uno de los mayores problemas en el diseño del cliente es el acceso a base de datos. Aunque la mayoría de los datos se encuentran en el servidor, existen datos en el cliente como son las sesiones de usuarios.

Para el acceso a este tipo de datos se ha desarrollado un pequeño ORM. Para el diseño de este componente se han creado una serie de anotaciones para definir los nombres de las tablas en las clases y los nombres de las columnas en los atributos. Podemos ver a continuación ambas anotaciones:

```java
package com.dbg.readycar.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * óAnotacin para relacionar una columna de SQLite con un atributo de un
    POJO.
 *
 * @author David Borrego
 * @since 19/04/2014
 */
@Target({ ElementType.FIELD })
@Retention(RetentionPolicy.RUNTIME)
public @interface ColumnName {

    /**
     * @return Nombre de la columna en bbdd
     */
    String name();
}
```

../readycar/ReadyCar/src/com/dbg/readycar/annotations/ColumnName.java

```java
package com.dbg.readycar.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * óAnotacin para relacionar el nombre de un POJO con el nombre de la tabla
     en
 * SQLite.
 *
 * @author David Borrego
 * @since 19/04/2014
 */
@Target({ ElementType.TYPE })
@Retention(RetentionPolicy.RUNTIME)
public @interface TableName {

    /** Nombre de la tabla */
    String name();
}
```

../readycar/ReadyCar/src/com/dbg/readycar/annotations/TableName.java

De esta forma ya tendríamos la configuración de las tablas.

Para el acceso a la base de datos se sigue el patrón de diseño Factory de forma similar a la empleada en el servidor. De esta forma con un DAO genérico tendría todas las operaciones necesarias únicamente heredando de este adaptador genérico:

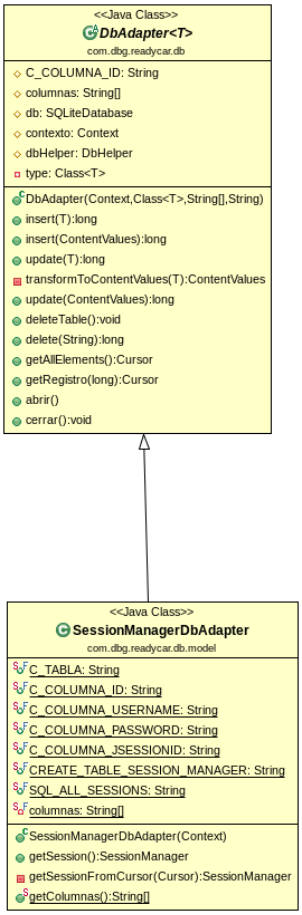

Figura 4.13: Diseño de la Capa de Integración en el Cliente

A la hora de trabajar con los datos de la aplicación no se trabaja con un modelo propio, se trabaja con los DTOs. El modelo únicamente se utiliza para operaciones propias del cliente, para todo aquello que tenga que ver con los datos de los usuarios se utilizan DTOs.

El motivo es sencillo: todos los datos están en el servidor por tanto las operaciones que realiza la capa de negocio de la aplicación es muy sencilla, únicamente realiza operaciones de petición, actualización, creación o eliminación de datos que vienen desde la vista o desde el servidor.

Para la comunicación con el servidor existe un paquete completo donde se encuentran centralizadas absolutamente todas las llamadas. Este paquete podría ser perfectamente un JAR a parte ya que es totalmente independiente.

En este paquete se encuentran una serie de paquetes, concretamente uno por entidad. Dentro de cada paquete existe una clase que implementa el API para esta entidad, pudiendo realizar cada una de las operaciones con los verbos correspondientes.

Para la comunicación se ha creado un HTTPClient propio de la arquitectura. Este HTTP Client es una adaptación del propio de Android. Las modificaciones realizadas giran en torno a la sesión del usuario. Es la propia entidad la que se encarga de gestionar el JSESSIONID, setearlo o levantar excepción en caso de que no se encuentre logueado el usuario.

De esta forma se simplifica el trabajo, es decir, a la hora de programar no hay que tener en cuenta nada relacionado con los usuario, la seguridad, etc ya que todo se encuentra implementado en cada una de las llamadas que forzosamente utilizan el HTTP Client.

Otro de los aspectos más importantes es cómo realiza internamente las llamadas según sean de tipo GET, POST, PUT o DELETE. Para esto se han creado cinco clases genéricas, una por llamada de las que sólo hay que extender y pasar los datos necesario. En caso de las llamadas de tipo GET se pueden realizar de forma síncrona o asíncrona.

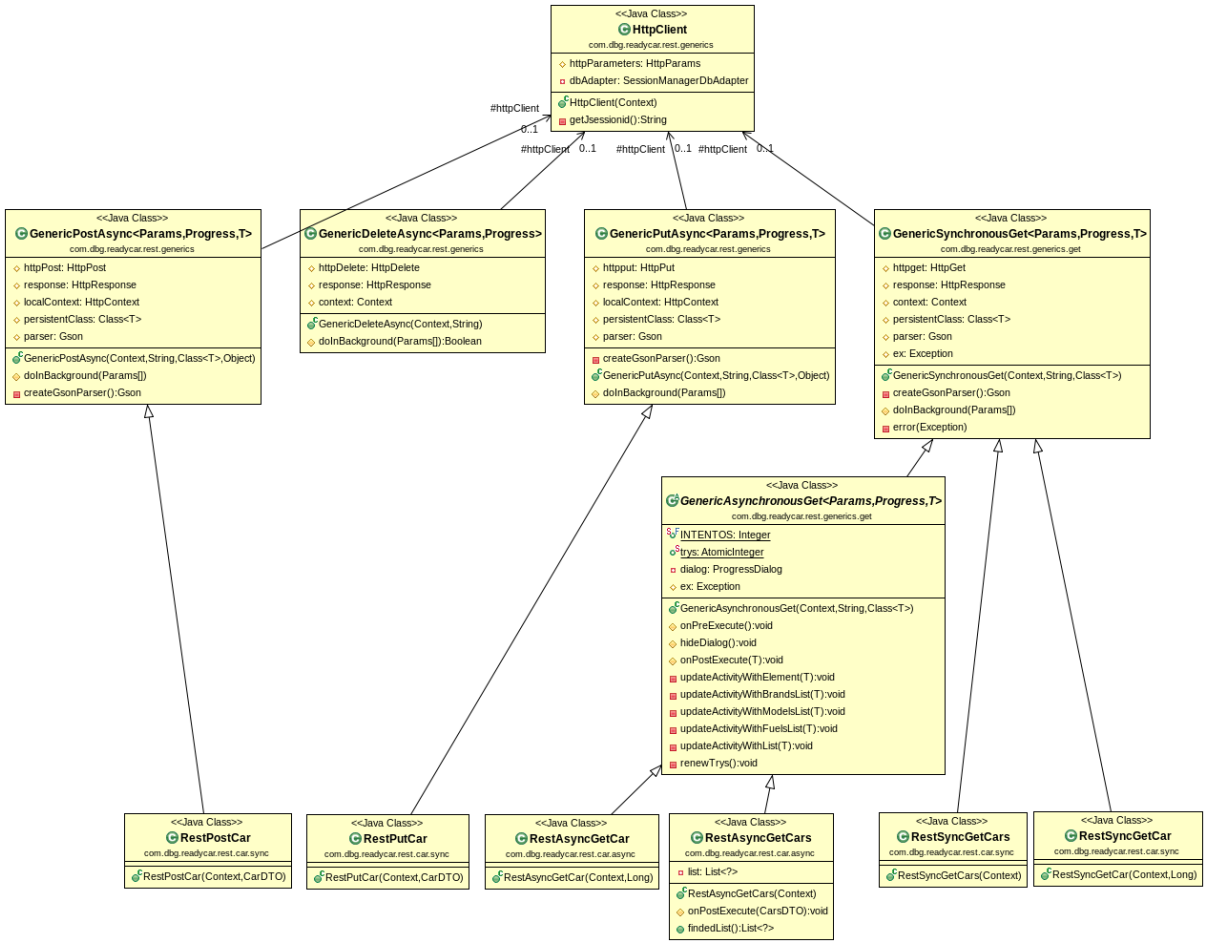

Figura 4.14: Diseño llamadas REST Cliente con ejemplo en «vehículos»

Otro punto a tener en cuenta es que en Android las llamadas son asíncronas. Esto puede ser una ventaja o en ocasiones una desventaja. Por este motivo las llamadas pueden realizarse de forma asíncrona o sincrona en el API.

La interfaz «CrudApi» es la encargada de definir todas las funciones que realizan las llamadas de cada una de las entidades. En la imagen anterior pudimos ver un ejemplo de utilización con el API para los vehículos, del mismo modo se utiliza en el resto de entidades.

Una vez implementada la interfaz, es la propia clase la que debe implementar todas las llamadas y encargarse de que las llamadas sean o no asíncronas.

De esta forma, cualquier programador que quisiera seguir evolucionando el sistema, ya tendría todas las llamadas implementadas y sólo tendría que hacer uso del API para realizar las operaciones en la vista. De esta forma se produce una abstracción completa y se simplifica el trabajo.

Para las llamadas de tipo GET, recuperación de datos, se ha implementado un sistema algo más complejo pero que permite abstraerse aun más. Gracias a una serie de interfaces que pueden implementar los presenter, es la propia llamada la que llama a estos servicios una vez ha recibido la respuesta.

Por ejemplo, si una pantalla con un listado realiza una petición para recibir los datos, su presenter debería implementar la interfaz anteriormente presentada. Esta interfaz posee la función «setElementList» y la función «refreshElements».

Una vez se ha recibido una respuesta correcta, se llama al método «setElementList» del presente que hubiera invocado la llamada REST. Inmediatamente después se llama al método «refreshElements» que es el método que se encarga de refrescar los elementos de la vista. De esta forma la programación es más sencilla ya que nos podemos abstraer de los mayores problemas surgidos en las llamadas.

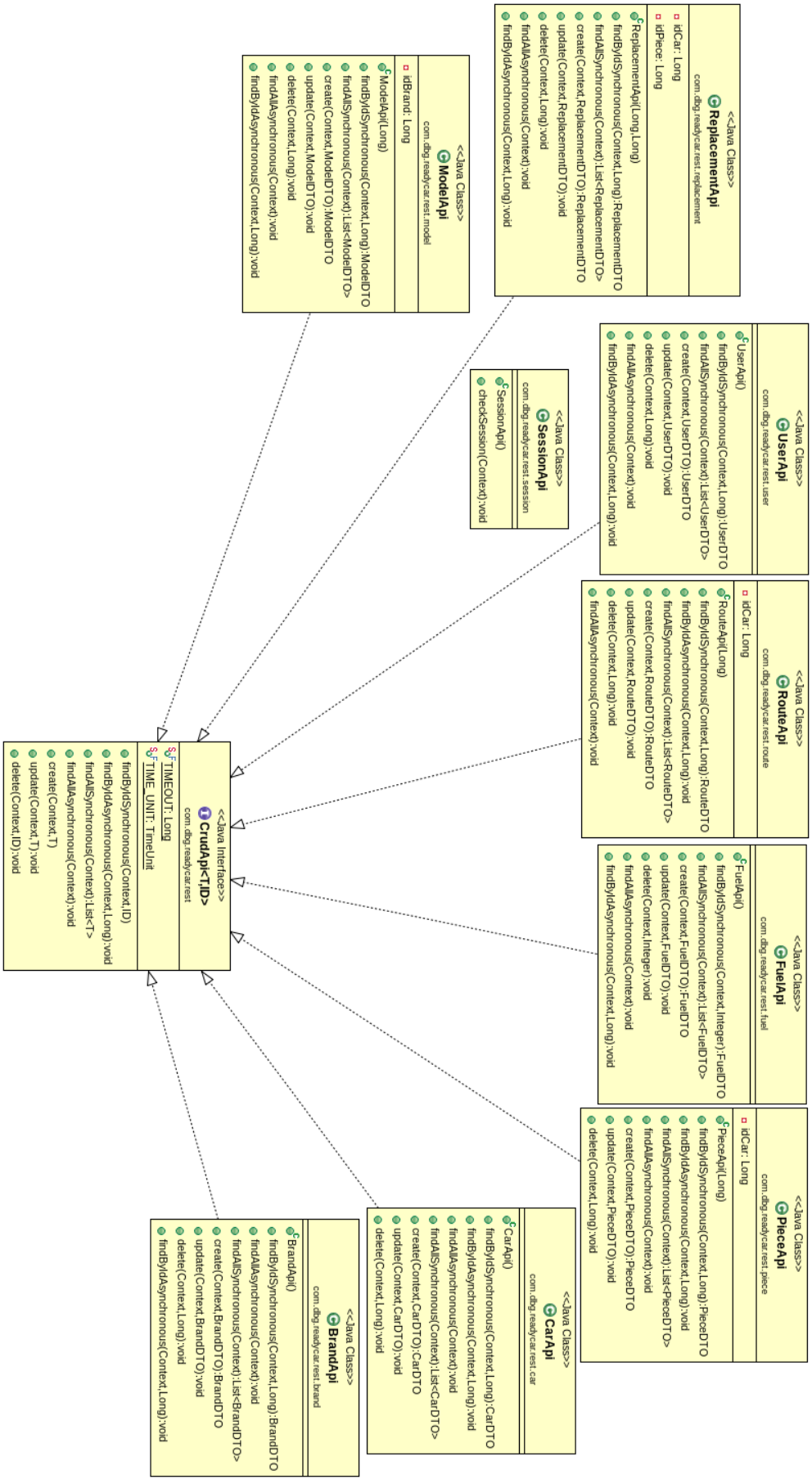

Figura 4.15: Diseño del API que realiza las llamadas desde el cliente.

Para el diseño de la vista se emplean los .xml clásico de Android. Este punto es uno de los puntos más claros del modelo MVP. Se crea la vista, con los componentes deseados en un fichero .xml totalmente independiente de la lógica de la pantalla.

Posteriormente, el presenter es el encargado de «dar vida» a este fichero o definición de la vista.

Para poder trabajar con el SDK de Android, uno de los aspectos más importantes es enteneder el ciclo de vida de un Activity. Este ciclo de vida se ve en la siguiente imagen:

Una vez entendido el ciclo de vida, sólo quedará realizar las peticiones, inicialización de componentes, recargas, etc en el momento adecuado.

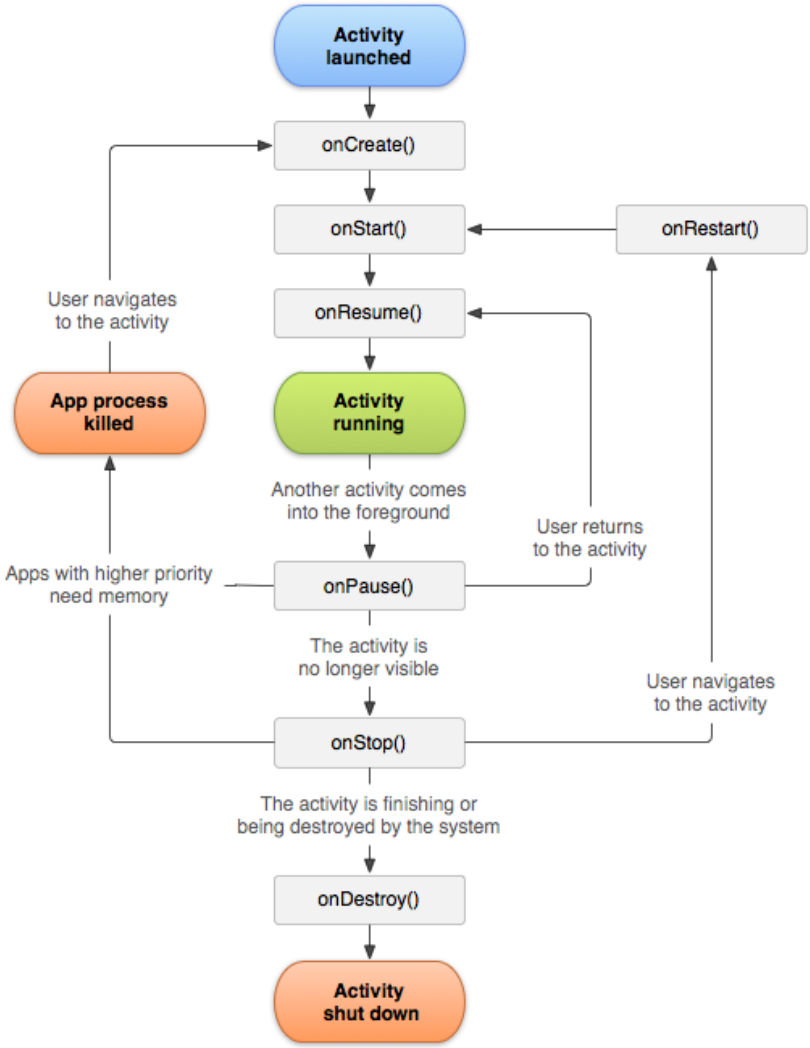

Figura 4.16: Ciclo de vida activity Android

De esta forma quedaría definido cada uno de los componentes del sistema, al menos a gran escala.

Capítulo 5

Implementación del Sistema

5.1. Entorno tecnológico

Como IDE he utilizado Eclipse. Comencé con la versión Kepler (4.3) y he terminado con la versión Luna (4.4). Utilizo Eclipse siempre que puedo, me parece un buen IDE, libre y con muchos plugin para la integración con casi cualquier tecnología.

Para el control de versiones he utilizado GIT. El control de versiones es uno de los puntos más importantes en cualquier proyecto. Ofrece la ventaja de tener un control sobre las versiones, tener el código a salvo de posibles perdidas y trabajar cómodamente en equipo.

Aquí podemos ver los primeros commit al proyecto:

Esta ha sido la forma de trabajar. He ido realizando ramas para nuevas funcionalidades y éstas las he ido «mergeando» con la rama master.

Otro punto importante es la integración continúa. Esto supone una automatización de las tareas y una mejora en el análisis del código. Para esta tarea he utilizado Jenkins, también descrito detenidamente en puntos anteriores y para la calidad de código SonarQube integrado en Jenkins.

En cuanto a lenguaje utilizado, frameworks, etc se puede ver un análisis detallado en el punto 4 de las presentes memorias.

5.2. Código fuente

Servidor

La aplicación del servidor ha sido creada como un proyecto Maven general. Dentro de este proyecto existen 3 módulos.

El primero de los módulos, «services-model», es el modelo de datos. Este módulo está a parte por si fuese necesario utilizarlo por otra aplicación.

El segundo módulo, «services-dto», es un JAR con los DTOs empleados en la aplicación. Este módulo se encuentra compartido con el cliente y como su nombre indica (Data Transference Objets) es empleado para la transferencia de datos y son los únicos objetos que salen o entran en la aplicación, es decir, son los objetos para cualquier tipo de comunicación con otra aplicación.

Por último, el módulo «services-rest» es el módulo que posee toda la lógica del sistema. Aquí es donde se encuentran las tres capas principales. Estas capas se encuentra divididas en paquetes, uno para cada capa. La capa de la vista (todos los servicios rest) se encuentran dentro del paquete «controller». Los DAOs se encuentran en el paquete «dao» y los servicios en el paquete «services». Dentro de cada paquete hay una paquete para cada parte de la aplicación, es decir, un paquete para todo lo relacionado con los vehículos, otro para lo relacionado con las rutas, etc.

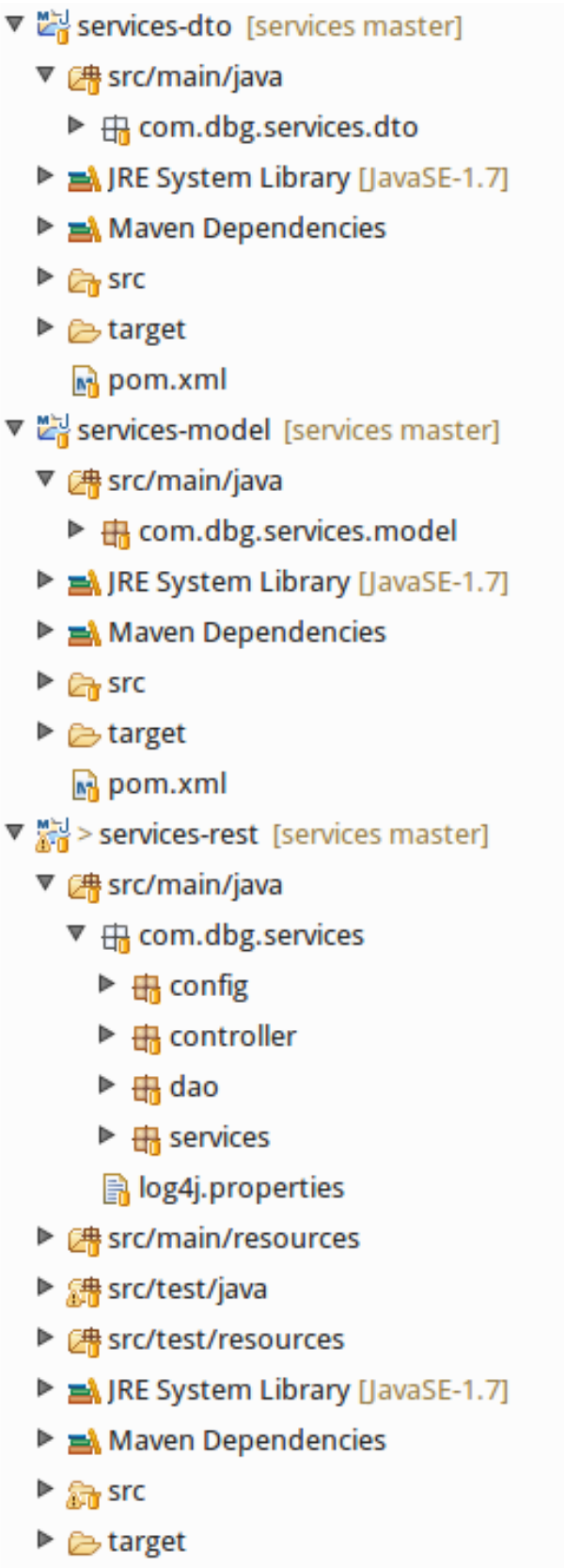

Figura 5.1: Estructura de paquetes Servidor

Cliente

El ciente ha sido estructurado dos proyectos, uno de compatibilidad generado automáticamente por Eclipse y otro con todo el contenido y la lógica de la aplicación.

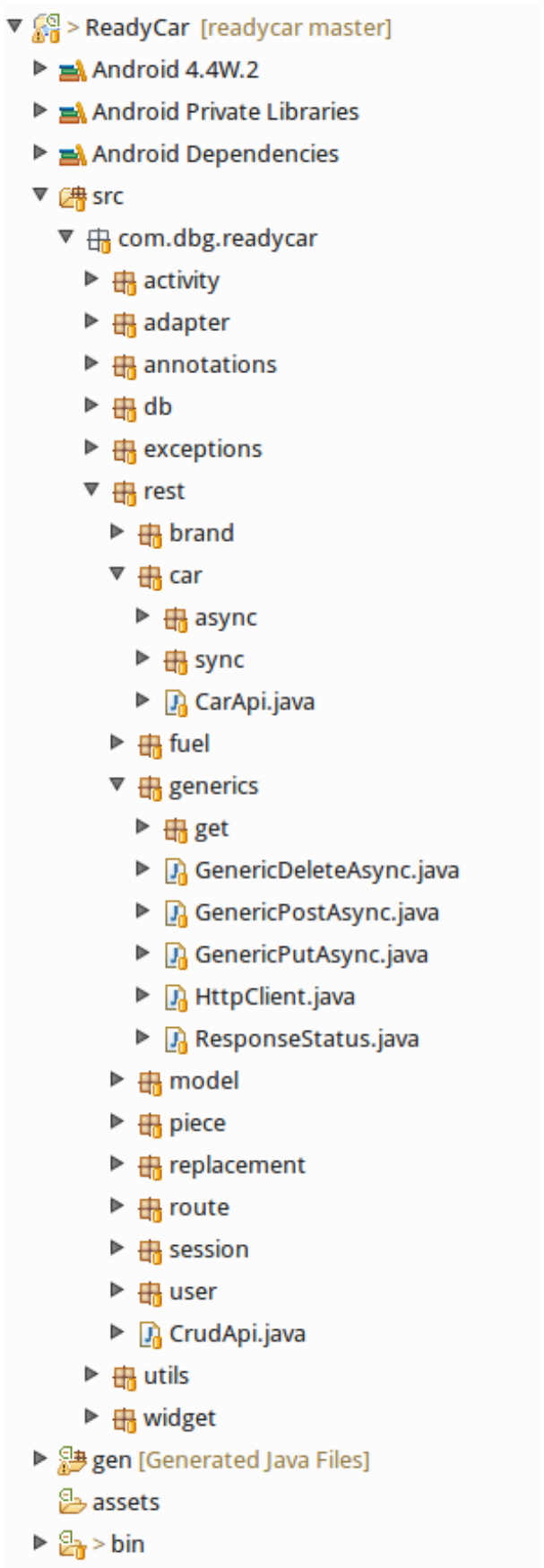

Figura 5.2: Estructura de paquetes Cliente

Podemos ver cada una de las funcionalidades de la aplicación en un paquete diferente. Dentro de cada paquete encontramos un paquete para cada tipo de contenido. Por ejemplo, en la siguiente imagen podemos ver como todo el api de llamadas «paquete rest» se encuentra abierto el paquete car.

De esta forma es muy sencillo encontrar cualquier contenido ya que vamos a la funcionalidad deseada y luego al tipo de contenido sobre el que se realiza dicha funcionalidad.

5.3. Calidad de código

La calidad de código es algo no muy tenido en cuenta en ocasiones. En mi opinión este punto es muy importante ya que el uso de este tipo de herramientas supone una mejora en el código y sobre todo en el programador.

Mi herramienta preferida para este tipo de labores es SonarQube. Esta herramienta es muy sencilla de utilizar y se integra perfectamente con Maven y Jenkins.

En primer lugar, procederé a la instalación del producto. El proceso de instalación es muy sencillo, sólo hay que bajarlo de su web `http://www.sonarqube.org/` y seguir los pasos de instalación, existen dos posibilidades: lanzar el script que ejecuta una versión con datos en memoria y servidor web embebido o realziar un .war mucho más consistente.

Una vez abierta la aplicación, es muy sencillo pasar los datos, unicamente hay que lanzar el comando «mvn sonar:sonar». Este comando se puede añadir a las tareas de Jenkins en cada uno de los depligues, en cada commit, etc.

Para poder ver más datos, no he realizado correcciones de código hasta una versión más avanzada del proyecto. El primer análisis realziado muestra los siguientes datos:

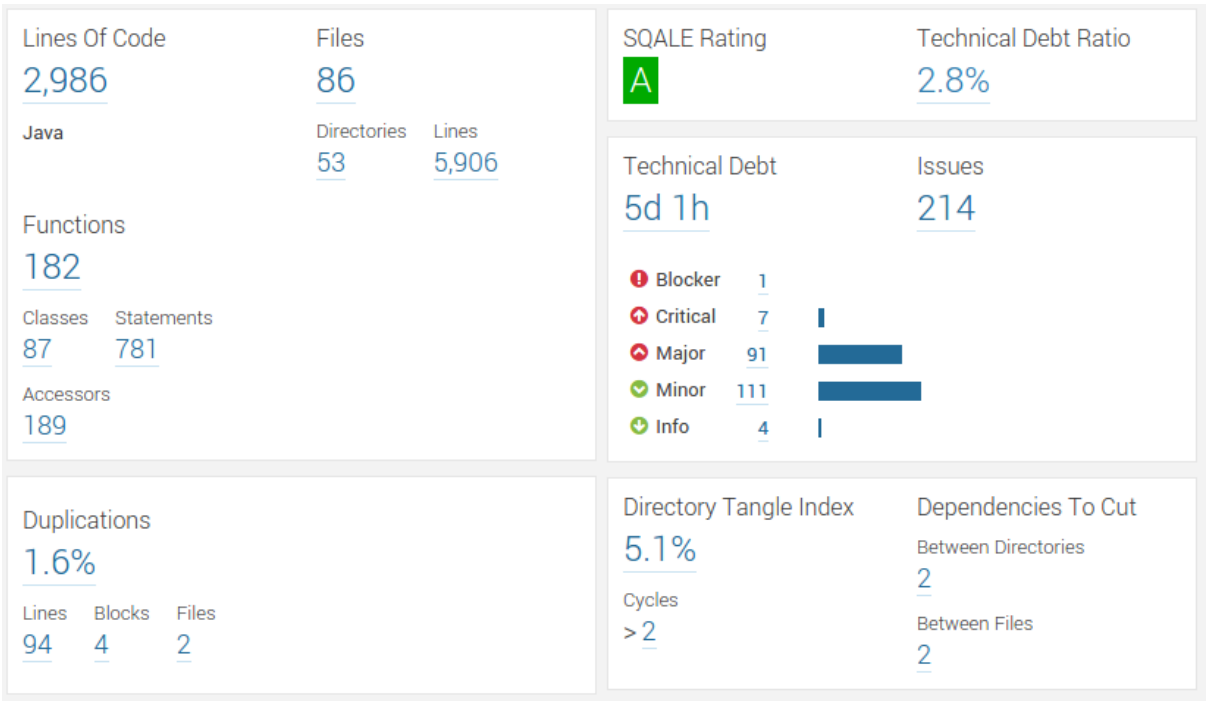

Figura 5.3: Primer análisis SonarQube

En las estadísticas podemos ver 2.986 líneas de código, 182 funciones, 86 ficheros, etc. Dentro del apartado de calidad podemos ver un SQALE Rating que es la calidad del código. El mayor valor es «A» y este es el valor que recibe el proyecto en el primer análisis.

De todos modos podemos ver las tareas de calidad de código a mejorar. Vemos una de tipo «Blocker», la más grave de todas, siete de tipo «Critical», noventa y una de tipo «Major» y cientonce de tipo «minor». La de tipo «minor» y «info» se muestran en verde ya que no son errores graves, incluso puede que no sean un error.

Otro punto muy interesante es la estimación de tiempo para solucionar estas deficiencias del código. Podemos ver una estimación de 5 días y 1 hora.

Si entramos en los errores, podemos ver detalladamente cuáles son, cómo solucionarlos, etc. En las siguientes imágenes podemos ver los errores de tipo «Critical» y «Blocker»:

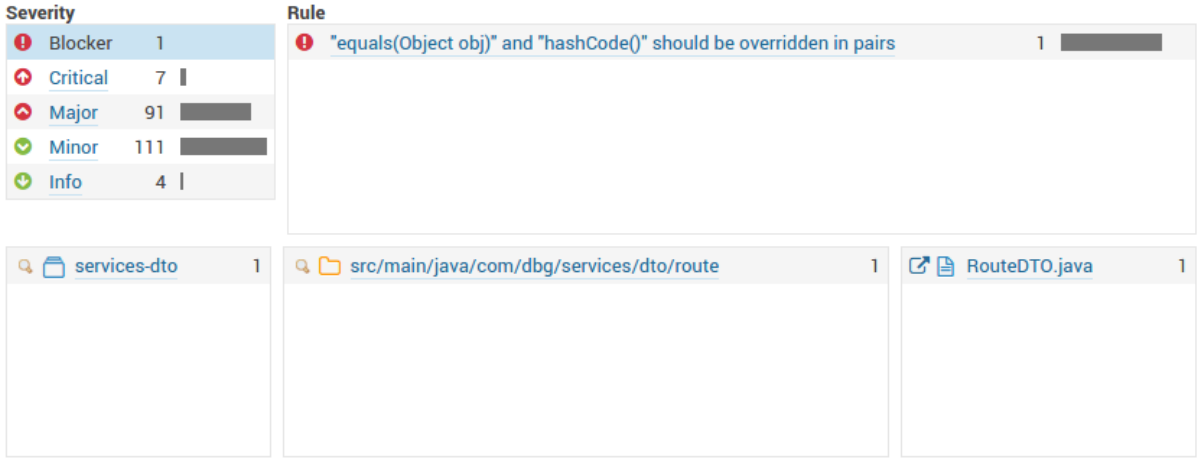

Figura 5.4: Calidad código: Blocker

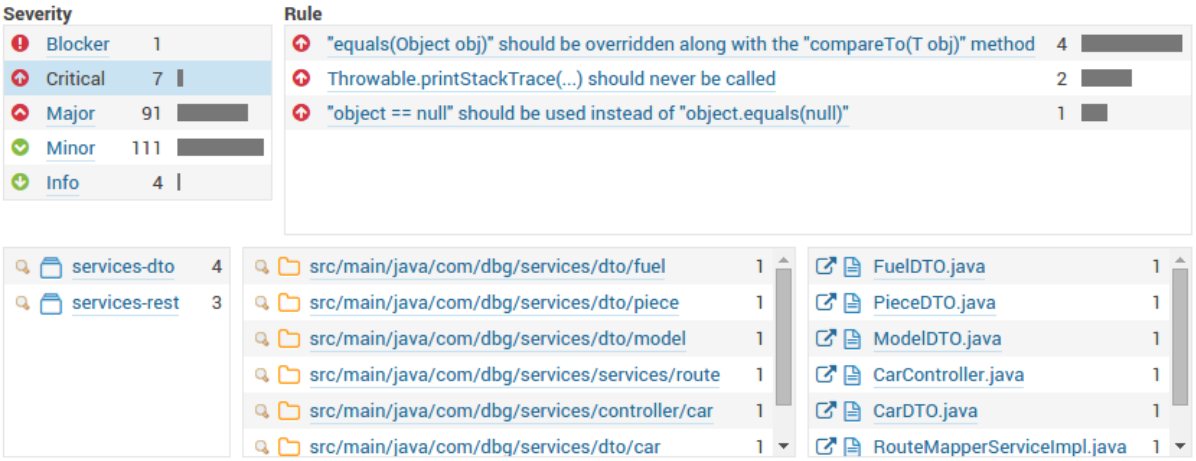

Figura 5.5: Calidad código: Critical

Tras ir revisando una a una todas las tareas indicadas por sonar llegamos al siguiente punto:

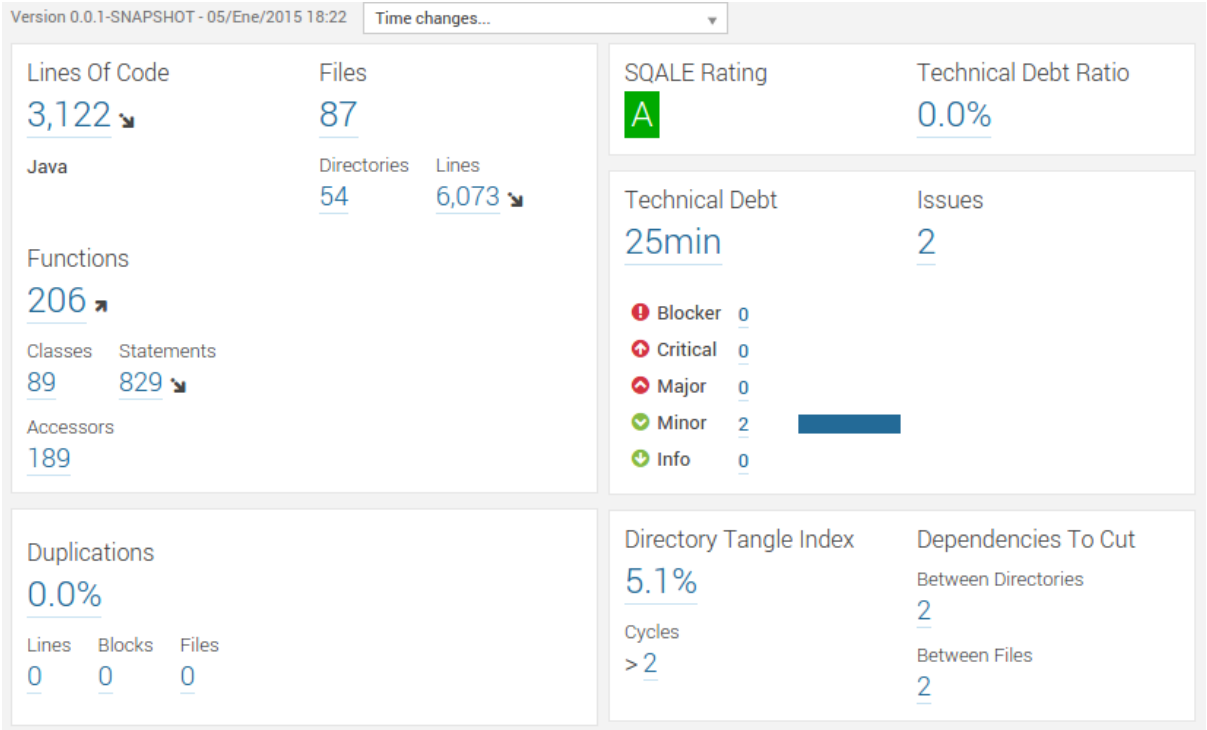

Figura 5.6: Segundo análisis de SonarQube con todo solucionado

Como podemos ver todos los errores han sido solucionados. Sólo quedan dos indicaciones de tipo «Minor». Las dos que quedan son indicaciones o recomendaciones. Exactamente son la recomendación de no declarar constantes en las interfaces y no utilizar números mágicos como índice de un enumerado:

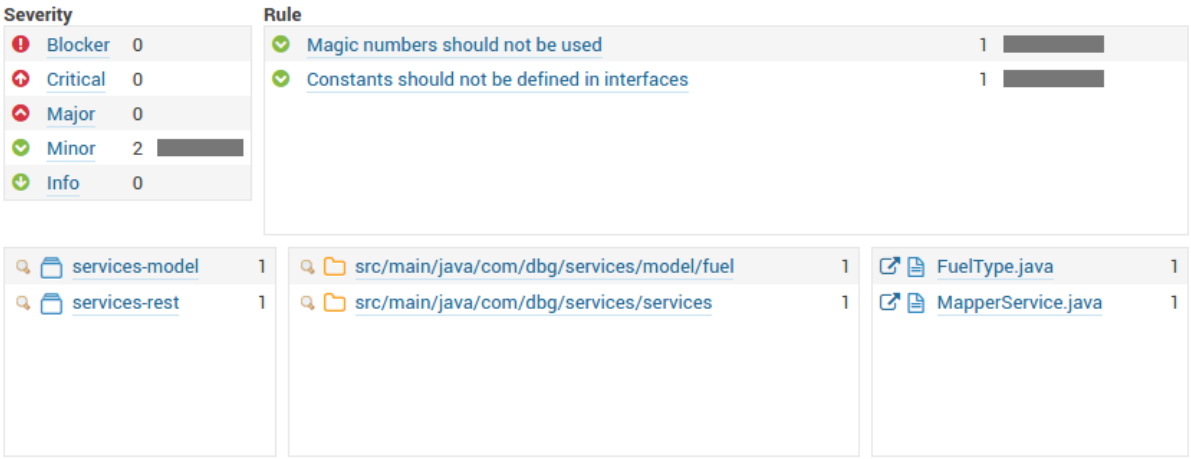

Figura 5.7: Solucionadas indicaciones de SonarQube

Otra de las ventajas que nos ofrece SonarQube es un análisis de la cobertura de código realizada con JUnit.

Para poder ver estos datos es necesario lanzar el siguiente comando: «mvn clean org.jacoco:jacoco-maven-plugin:prepare-agent install -Dmaven.test.failure.ignore=true && mvn sonar:sonar». Al igual que el otro este comando se puede lanzar desde Jenkins o a mano.

Para tener una cobertura de código aceptable, la mayoría de autores hablan de un rago de entre 60 % y 70 % dependiendo de la cantidad de POJOs o clases simples que tengamos en nuestro sistema. Por ejemplo: al tener un paquete de DTOs sin lógica ninguna no es necesario realizar test para analizar todos los getters y setters ya que no va a aportar nada al proyecto en relación al tiempo empleado en esta labor.

Para mi proyecto entiendo que es necesario un rango mucho más alto. Esto se debe a que tengo un módulo para el modelo, otro módulo para los DTOs y por tanto el módulo con el negocio de la aplicación posee toda la lógica y es aquí donde interesa tener el mayor rango de cobertura posible.

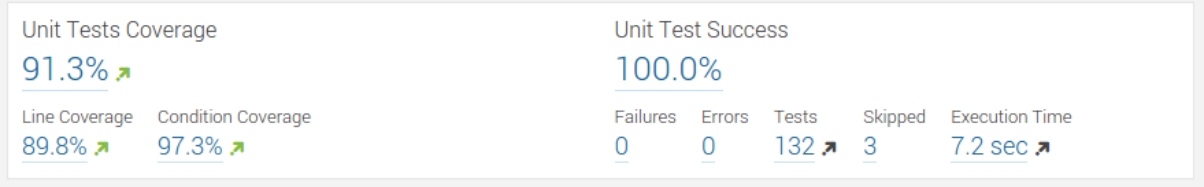

Figura 5.8: JUnit: Análisis de cobertura de código

Podemos ver una cobertura de código del 91.3 % y un 97.3 % de las condiciones de la aplicación cubiertas. Por otro lado, vemos que esto se consigue gracias a 132 test unitarios ejecutados en 7.2 segundos.

Capítulo 6

Pruebas del Sistema

En el apartado anterior ya se ha hablado detenidamente a cerca de los test y la calidad del código. En este apartado se puede ver una visión más técnica, un detalle a las tecnologías y técnicas empleadas para testear el sistema.

En primer lugar se puede encontrar las pruebas unitarias, su función y el porqué de utilizarlas.

Posteriormente se habla a cerca de las pruebas de integración del sistema, como conviven cliente y servidor y cómo se realizan todas las pruebas de integración que hacen que el sistema sea más robusto.

6.1. Pruebas unitarias

La pruebas unitarias nos proporcionan un testeo de nuestro código de forma totalmente independiente a otras capas o servicios. Por ejemplo, si estamos probando un servicio que accede a otro sistema, el test no debería llamar a ese sistema ya que la prueba quedaría condicionada a la respuesta.

Otro aspecto muy importante para realizar pruebas unitarias es el utilizar Mocks. Un mock es una forma de simular llamadas o respuestas. Cuando necesitamos realizar una llamada en un test, esta llamada se puede «mockear» y ser simulada de forma que siempre obtendremos la misma respuesta y nuestra prueba no se verá afectada.

Para las pruebas del Servidor, he utilizado [JUnit(2014)]. Esta libreria permite realizar test unitarios de forma rápida y sencilla. Otra de las posibilidades que nos ofrece es el realizar pruebas de integración ya que se puede simular toda una llamada entre capas.

Para la realización de los Mocks he utilizado [Mockito(2014)]. Además de mejorar la velocidad de testeo y facilitar el trabajo mockito proporciona una fácil interfaz. Es tan sencillo de utilizar como decir «cuando se realice una petición pidiendo la ruta con id 1, devuelve esta ruta». De esta forma se simplifican las labores de testeo.

Una desventaja de mockito es que no es posible relizar testeos de método estáticos o privados. En caso de que fuese necesario se puede utilizar «power mockito» para esta labor.

Al igual que el servidor, la parte cliente también puede ser testeada con pruebas unitarias. Concretamente he utilizado [Robotium(2014)].

Robotium es un framework muy sencillo de utilizar, tan sencillo que el construye los test de forma automática. Para realizar un test basta con arrancar la aplicación en un dispositivo móvil y pulsar sobre el boton «grabar» que nos proporciona el propio framework. Cada clic o acción que realizamos en la aplicación se transforma a uno orden en el test. Posteriormente sólo tendríamos que añadir las comprobaciones deseadas al test y ya tendríamos una forma de repetir todos los movimientos realizados a través de la aplicación de forma automática.

Otra aspecto a tener en cuenta es que al testear la parte servidora, también se esta asegurando la integridad de los datos en la parte cliente. Si a esto le añadimos test de integración tendremos una posibilidad de error mínima.

Se puede ver más a cerca de todas las pruebas unitarias realizadas, cobertura e integración continua en el apartado 5.3

6.2. Pruebas de integración

Las pruenas de integración realizadas han sido de dos tipos. En primer lugar se ha comprobado que en la parte servidora todo se encuentra de forma correcta a través de test que simulan las llamadas y llegan hasta base de datos, siguiendo todo el proceso normal de un usuario haciendo uso de la aplicación.

Para la realización de estos test ha sido creado un entorno de pruebas alternativo, similar al de producción y con datos válidos y extremos para la realización de pruebas.

Otras pruebas de integración ha sido la realización de grabaciones con Robotium comprobando que el flujo de navegación entre las pantallas se cumple, es decir, partiendo desde la pantalla principal se puede navegar de forma correcta hasta cualquiera de las pantallas más «alejadas». Para este tipo de navegación automática se realizan llamadas al servidor de forma idéntica a la que realizaría un usuario.

Este tipo de pruebas han sido grabadas y se ejecutan para cada una de las pantallas del sistema.

6.3. Pruebas de aceptación

Las pruebas de aceptación tienen como objetivo verificar que la aplicación se encuentra estable para un paso a producción.

Las pruebas de aceptación para este sistema se componen de todos los test unitarios, más los test de integración más pruebas manuales que se realizan simulando la interacción de un usuario.

Todo esto hace que el sistema disponga de más de 100 test para comprobar de forma automática que todo se encuentra correcto.

Capítulo 7

Manual de usuario

A continuación podemos ver las instrucciones de uso del sistema. Puesto que el servidor sólo se dedica a realizar las peticiones que recibe, su existencia es indiferente a un usuario, por tanto este manual de usuario sólo es para entender el uso de la aplicación móvil.

7.1. Características

El funcionamiento del sistema se presenta sencillo, con pantallas intuitivas y cuidando al máximo la usabilidad.

En primer lugar nos encontramos con una pantalla inicial. Esta pantalla contiene el menú que lleva hasta las pantallas distributivas de cada una de las funcionalidades de la aplicación. Por ejemplo, desde la pantalla principal se puede ir a la pantalla distributiva de vehículos y posteriormente realizar las operaciones deseadas con cada uno de los vehículos.

Otro aspecto muy importante es el registro. Aunque es un registro instantáneo y sencillo, éste es necesario para poder utilizar la aplicación. Esto se debe a que la información almacenada tiene que ser securizada.

7.2. Requisitos previos

Sólo es necesario disponer de un dispositivo móvil con Android y conexión a internet.

7.3. Utilización

A continuación detallo cada una de las pantallas principales de la aplicación.

7.3.1. Pantalla principal

La pantalla principal se compone de una cabecera, un cuerpo para cualquier tipo de contenido y un pie. En la cabecera podemos ver el nombre de usuario registrado, un «spinner» para la selección del vehículo activo y un botón para hacer «logout». Todo esto se ve así cuando el usuario se encuentra logueado:

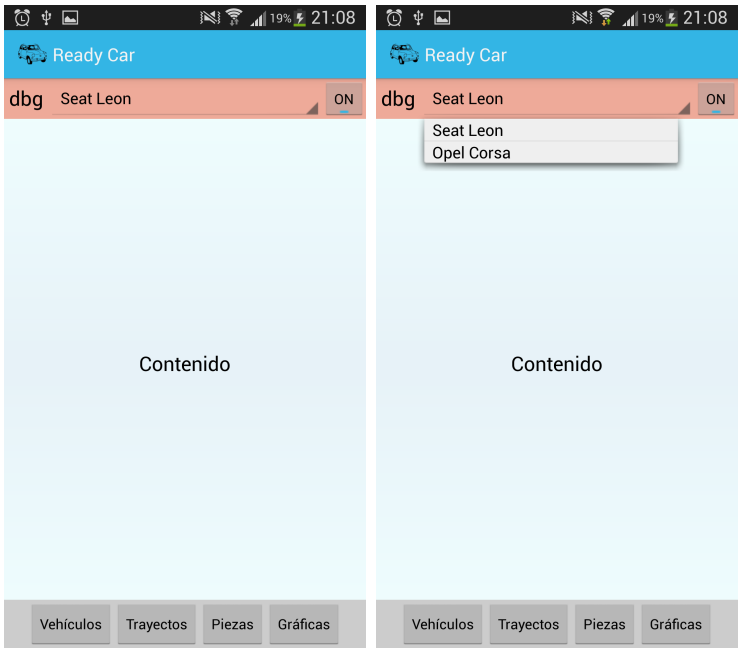

Figura 7.1: Pantalla principal de un usuario logueado

En la primera de las pantallas se ve un usuario «dbg» logueado y un spinner con sus vehículos, el cuál se encuentra desplegado en la segunda de las imágenes. Este «spinner» tiene gran importancia ya que todas las operaciones realizadas en el resto de pantallas se realizarán sobre este vehículo, es decir, el vehículo que se encuentre seleccionado en la pantalla principal.

En caso de que el usuario no se encuentre logueado la cabecera de la pantalla cambiará. En ese caso, se sustituirá el «spinner» por el botón de «Registro» y el menu del pie quedara deshabilitado. Podemos ver a continuación este caso:

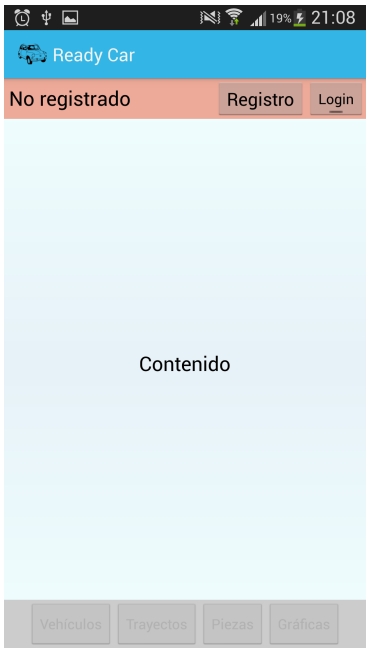

Figura 7.2: Pantalla principal de un usuario anónimo

7.3.2. Vehículos

En primer lugar podemos ver las pantallas distributivas de vehículos. Desde esta se puede crear un nuevo vehículo o editar uno existente. Además vemos el logo de la marca, el nombre del modelo y los kilometros realizados.

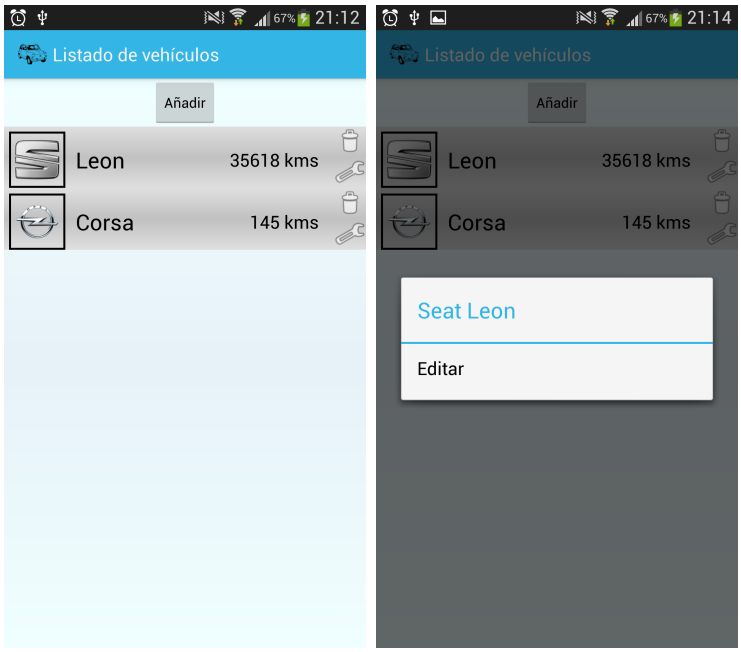

Figura 7.3: Pantalla distributiva de vehículos

A continuación, una vez elegida la opción de añadir vehículo o modificar uno existente se mostraría una de las siguientes pantallas:

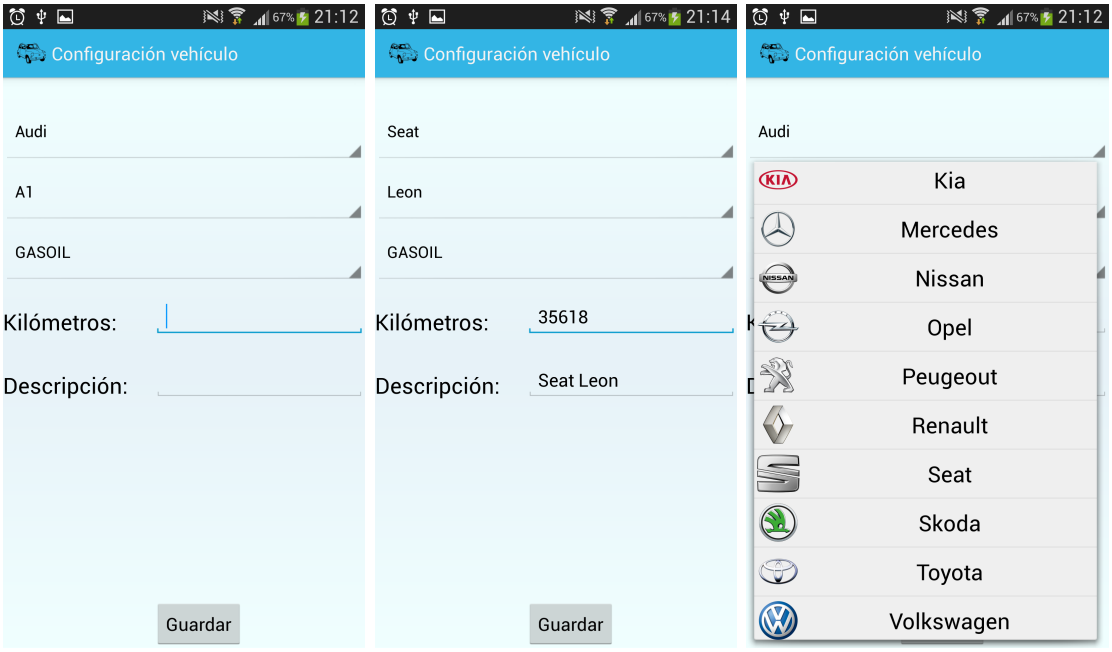

Figura 7.4: Pantalla para añadir o modificar un vehículo

La primera de las pantallas es la pantalla para añadir un nuevo vehículo. Para realizar esta operación sólo es necesario elegir la marca, el modelo, el combustible, los kilometros iniciales y una descripción en caso de que se desee.

La segunda pantalla es la patalla iniciada con los datos para modificar un vehículo existente. En esta pantalla sólo sería necesario modificar los datos deseado y guardar el vehículo para que los cambios se apliquen.

La tercera pantalla es un ejemplo de selección de marca, una vez elegida la marca del vehículo se produce la carga de sus modelos, de forma que sólo veremos los modelos de la marca seleccionada.

7.3.3. Trayectos

Los trayectos se muestran en un listado ordenable. Se puede ordenar por fecha, días, consumo o kilometros tanto de forma ascendente o descendente. En el listado de trayectos también se puede ver el ranking con el top 3 de los trayectos con menor consumo a los 100 kilometros. Otro punto a tener en cuenta es el color del consumo, en caso de que este sea mayor a la media del vehículo se mostrará en rojo, encaso contrario en verde.

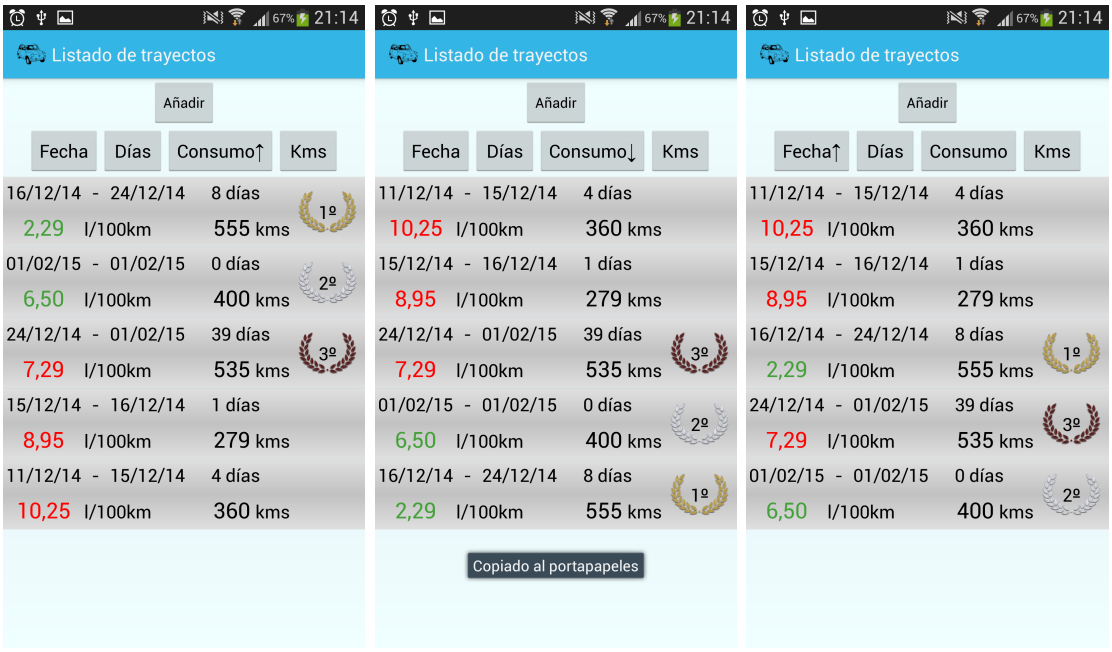

Figura 7.5: Pantalla distributiva de rutas

En la primera imagen se muestran las rutas ordenadas por consumo en orden ascendente, en la segunda por orden descendente y en la última por fecha de inicio de forma ascendente.

Desde esta pantalla se puede añadir una nueva ruta, modificar una ya existente, eliminar una no deseada o ver detalladamente cualquiera de las rutas. Haciendo clic sobre una de las rutas se muestra el siguiente diálogo:

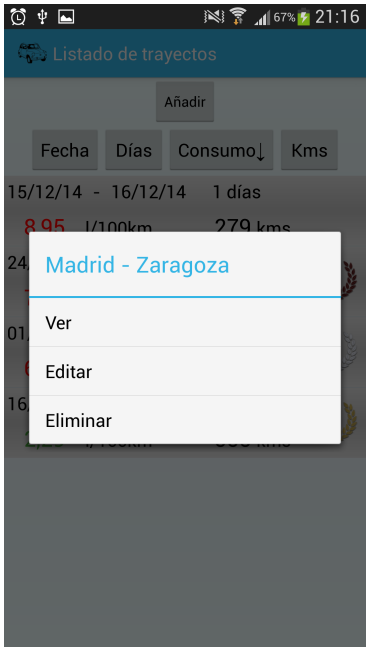

Figura 7.6: Pantalla distributiva de rutas con menu de selección

En caso de ser seleccionada la opción de añadir, editar o ver se muestra la siguiente pantalla:

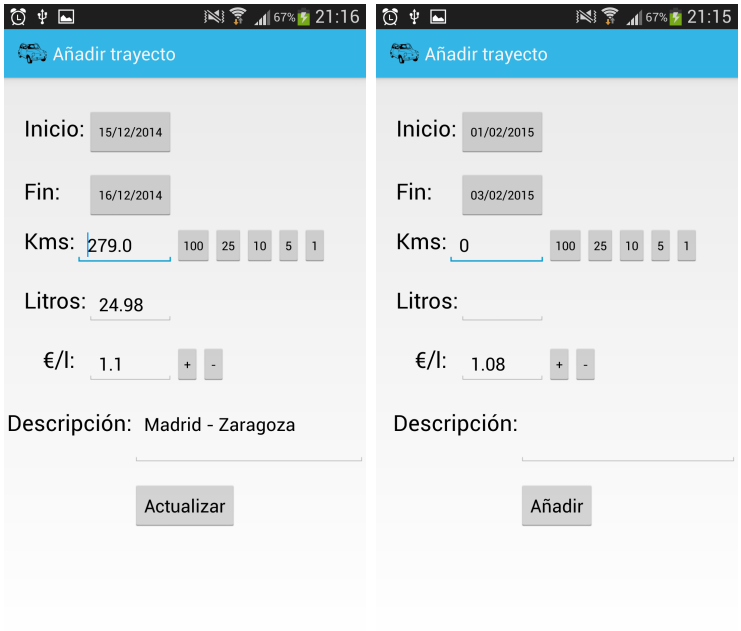

Figura 7.7: Pantalla para añadir o editar ruta

La primera de las imágenes corresponde a la pantalla de edición de ruta, como se ve la pantalla se inicializa con los datos de la ruta, una vez modificado los datos deseados sólo será necesario pulsar en el botón «Actualizar» para que los cambios queden registrados.

En la segunda imagen se puede ver la opción de añadir una nueva ruta. Esta pantalla pretende ser lo más sencilla posible y por este motivo el campo «Inicio» se encuentra inicializado con la fecha de fin del último trayecto y el campo de «Fin» con la fecha del día en cuestión. El campo kms puede ser incrementado pulsando en los números a la derecha. Para el precio ya se inicializa con el último precio del combustible registrado en el sistema.

Una vez pulsado en «Añadir» o «Actualizado» el sistema vuelve a la pantalla distributiva de rutas con los datos actualizados.

7.3.4. Piezas

Las piezas se muestran ordenadas en un listado, la siguiente pantalla representa todas las piezas de un vehículo determinado:

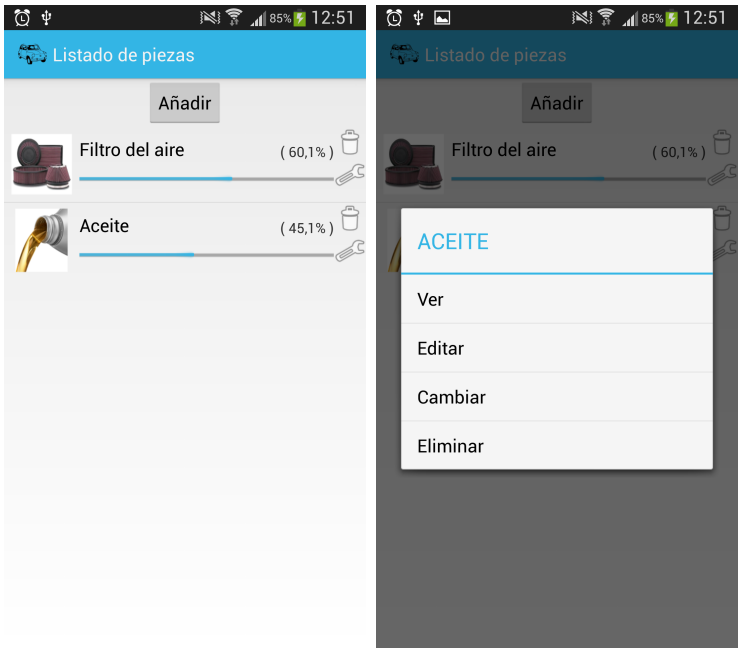

Figura 7.8: Pantalla distributiva de piezas

En la primera pantalla se ve el listado de pieza. Desde pantalla es posible añadir una nueva pieza o realizar las operaciones deseadas sobre una en cuestión. Al pulsar una de las piezas se abre el menu tal y como podemos ver en la segunda pantalla.

Al pulsar sobre el la opción de «Eliminar», la pieza será eliminada.

Si se pulsa sobre «Editar» o sobre «Ver» se abre la siguiente pantalla:

Figura 7.9: Pantalla distributiva de piezas

En esta pantalla se puede ver un listado de los cambios realizados sobre la pieza. Una carita que dependiendo del color indicará si el cambio se ha hecho cumpliendo las especificaciones de vida de la pieza o no. También podemos ver una serie de estadísticas referentes al coste de la pieza.

Otra de las opciones del menu corresponde al cambio de una pieza, si se pulsa sobre esta opción se procede a indicar un cambio, es decir, se ha realizado un mantenimiento a esta pieza. Para esta opción se muestra la siguiente pantalla:

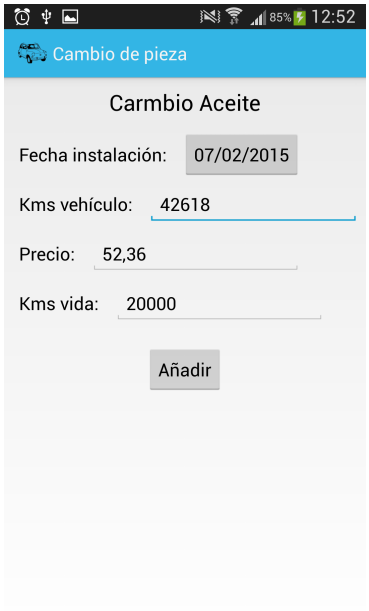

Figura 7.10: Pantalla para la sustitución de una pieza

Al hacer clic sobre añadir se cargará la siguiente pantalla:

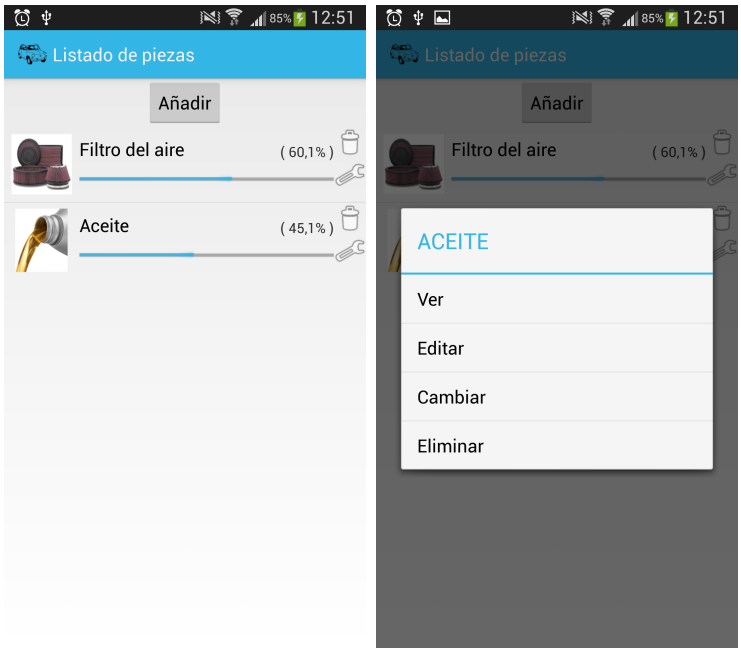

Figura 7.11: Pantalla para añadir una pieza

En este caso se añadirá una pieza al vehículo indicado.

7.3.5. Estadísticas

La pantalla de estadísticas muestra un gráfico con el precio del combustible desde el primer trayecto registrado en el sistema hasta el último.

En el siguiente ejemplo podemos ver datos reales del precio del díesel desde diciembre de 2013 hasta enero de 2015.

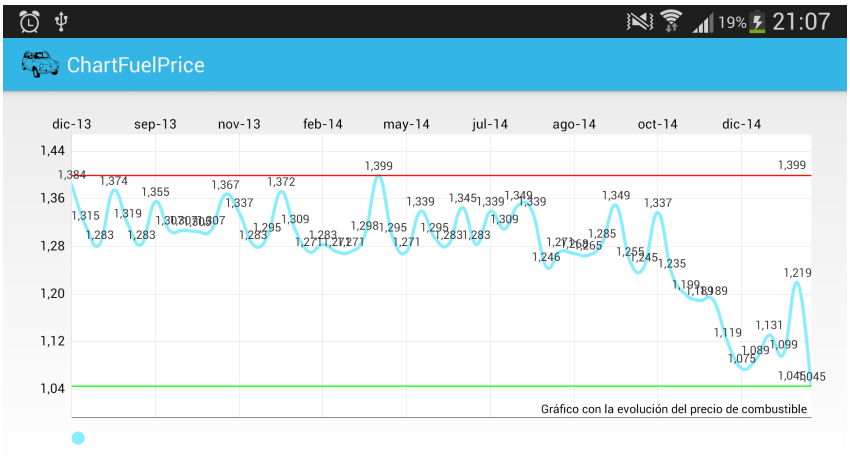

Figura 7.12: Pantalla con gráfica precio combustible desde enero 2.013 hasta enero 2.015

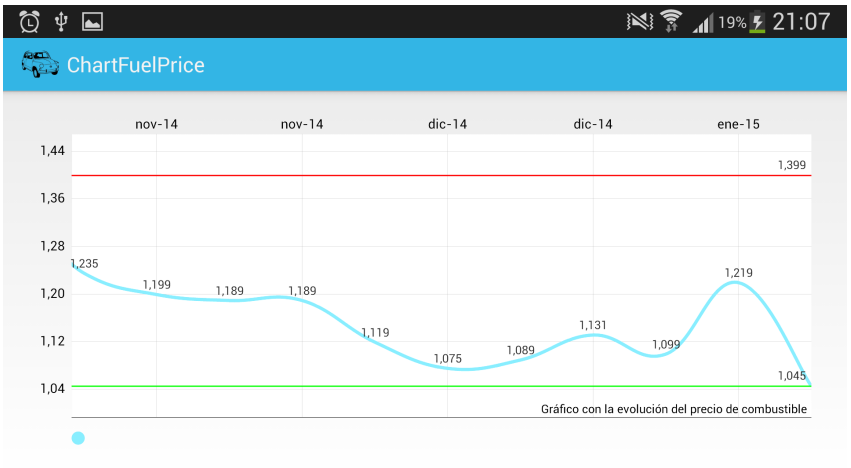

Figura 7.13: Pantalla con gráfica precio combustible desde 10/14 hasta 01/15

7.3.6. Usuarios

La primera operación que debe realizar un nuevo usuario es el registro. Esta operación es extremadamente sencilla ya que sólo hay que rellenar la siguiente pantalla:

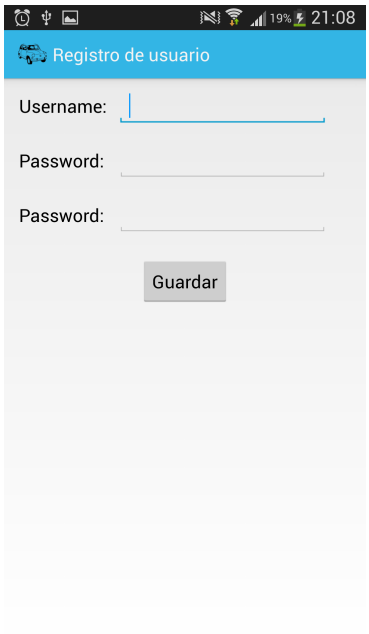

Figura 7.14: Pantalla para el registro de un nuevo usuario

Una vez registrado el usuario ya podrá realizar el «login» en la aplicación cada vez que lo desee utilizando sus credenciales. Esta operación se realiza a través de la siguiente pantalla:

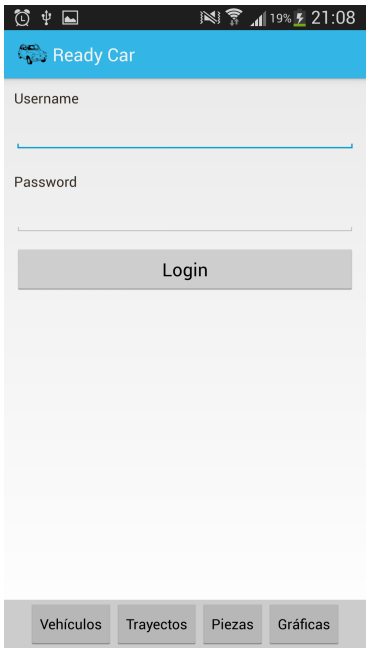

Figura 7.15: Pantalla para el login de un usuario

Una vez logueado el usuario sus datos serán guardados por el sistema y no será necesario realizarlo más si no se desea.

Capítulo 8

Manual de instalación y explotación

En esta sección son detallados los pasos a seguir para poder poner en producción el sistema. Únicamente se habla de la configuración e instalación del sistema con el servidor y el cliente realizando peticiones contra ese servidor.

De forma más detallada se puede ver el apartado 4.1.2 donde se detalla el sistema actual de producción, con un cluster, replica de sesión, etc.

En primer lugar se habla a cerca de los requisitos necesarios tantos hadware como software.

Posteriormente se indica los pasos a seguir para configurar el entorno de producción, su instalación en un servidor de aplicaciones y las modificaciones necesarias para poder configurar el cliente.

8.1. Requisitos previos

Como requisitos previos son necesarios al menos tres servidores para asegurar la alta disponibilidad. Recomiendo la utilizanción de cualquier distribución de Linux sin interfaz gráfica o con una interfaz gráfica mínima como por ejemplo Xubuntu.

Es importante aclarar que la utilización de tres nodos es únicamente para temas de alta disponibilidad, en caso de no disponer de tantos servidores sería suficiente con utilizar un servidor con todas las tecnologias necesarias.

La aplicación servidora necesita conexión a un MySQL o cualquier otra base de datos relacional que conecte con Hibernate de forma correcta. Una máquina virtual de Java 1.7 y un servidor de aplicaciones, a ser posible Tomcat 7.

Para poder utilizar el cliente sólo necesitaremos un dispositivo Android con conexión a internet.

8.2. Inventario de componentes

Los componentes serán un fichero .WAR con la aplicación servidora y una APP para Android.

8.3. Procedimientos de instalación

El WAR contiene la aplicación preparada para un entorno de producción. Sólo es necesario indicar dónde se encuentra el MySQL para arrancar. Esto se indica en el fichero database.properties. Aquí habrá que indicar el servidor, el esquema y el usuario y contraseña.

Una vez configurado el .WAR sólo es necesario arrancar en un servidor de aplicaciones, a ser posible Tomcat 7 con la raiz de contexto en «/».

Para tener la alta disponibilidad necesitamos otro servidor que realice las funciones de balanceador. Esto se consigue gracias a un Apache con el módulo jk.

Para que el cliente funcione contra esta nueva instalación sólo es necesario modificar la clase «Constants.java» y poner la ip o el dominio de esta nueva instalación.

En sistema actual de producción lo que se ha hecho es adquirir un dominio de forma que aunque cambie la ip, el dominio siempre será el mismo.

8.4. Pruebas de implantación

Debido a la gran cantidad de pruebas unitarias e integración que tenemos en el sistema las aplicaciones estarías probadas. Sólo sería necesario probar la conexión entre cliente y servidor. Esto se produce al abrir la aplicación e intentar realizar el login. Si este punto funciona todo estaría correcto ya que para esta función se realiza una conexión entre cliente, servidor y MySQL.

Capítulo 9

Conclusiones

En esta sección hablaré de todo lo aprendido a base de errores y aciertos. Haré especial incapié en todo aquello que considero un acierto y me aportará un plus a lo largo de toda mi carrera profesional.

Posteriormente, haré un repaso a toda la posible evolución del proyecto, evaluando diversas posibilidades y abriendo diferentes posibilidades de evolución.

9.1. Objetivos

Creo que los objetivos del sistema eran muy ambiciosos, de forma general, éstos han sido cumplidos ampliamente.

Como parte no funcional he conseguido un sistema bastante estable. Los objetivos no funcionales han sido cumplidos en cuanto a seguridad, escalabilidad, fiabilidad y alta disponibiblidad. La plataforma es bastante estable y su coste tanto de instalación como mantenimiento es muy ajustado.

Otro aspecto no funcional era el rendimiento, el tiempo de respuesta se encuentra en la mayoría de los casos por debajo de un segundo. Este objetivo creo que es bastante mejorable y con una plataforma de servidores más potentes el tiempo de respuesta se reduce drásticamente.

En cuanto a la parte funcional, nos encontramos con un sistema que cumple todos los aspectos deseados desde la parte del Servidor. En la parte del cliente nos encontramos con un sistema totalmente integrado.

Este aspecto es donde los objetivos podrían haber sido cubiertos más ampliamente. Utilizando otro sistema para la programación del cliente, con el mismo esfuerzo podría haber obtenido un impacto en un mayor número de dispositivos. Por ejemplo, utilizando algún modelo JavaScript con una versión responsive hubiera tenido un sistema web que haciendo uso de los mismos servicios REST hubiera podido ser utilizado por dispositivos Android, IPhone, PCs, Tablets, etc.

El objetivo más importante para mi ha sido el aprendizaje y es en este aspecto donde más satisfecho me encuentro. Durante todo el desarrollo he ido realizando pruebas de conceptos con las tecnologías empleadas y otras que he decidido no utilizar, esto me ha permitido conocer un gran abanico de Frameworks, lenguajes y posibilidades.

9.2. Lecciones aprendidas

Creo que el PFC me ha servido como punto de encuentro de todos los conocimiento adquiridos durante toda la titulación. A parte, he conseguido crear una aplicación con un tanto de complejidad, verla funcionar y poder enfrentarme con éxito a los diferentes problemas surgidos durante el desarrollo.

Uno de los puntos más importantes aprendido a lo largo de toda mi carrera profesional y del presente proyecto es el trabajar con una arquitecura propia. Esta ha sido la forma

de trabajar durante todo el PFC, me he centrado en crear una arquitectura lo más óptima posible y posteriormente hacer uso de ella. En la mayoría de los casos esto supone un avance más lento en fases iniciales del proyecto pero a medio y largo plazo supone un punto extra. De esta forma tendríamos las siguiente ventajas entre otras:

- Evita repetición de código.
- Centramos los puntos de complejidad y error en clases abstractas de forma que podemos centrarnos en optimizar estos puntos.
- Proporcionamos únicamente los datos y operaciones deseadas.
- Dejamos el modelo para la arquitectura y los DTOs para posibles integraciones, ocultando nuestro modelo.
- La arquitectura puede ser más fácilmente probada y controlada por las mismas personas, reduciendo la posibilidad de error.
- Claridad.

Otro punto muy importante es el uso de un API. Esto supone algo muy importante por varios motivos:

- Se centralizan todas las llamadas, evitando duplicidad de código.
- Se puede controlar fácilmente las llamadas que se realizan, tanto a nivel de seguridad o estadístico.
- Se puede proporcionar a terceros. Es decir, haciendo uso de DTOs, un tercero pordría hacer uso de nuestro sistema tal y como nuestro API se lo facilite sin conocer absolutamente nada de nuestra arquitectura.

He estudiado y entendido la utilidad y ventaja de conocer los diferentes patrones de diseño, dónde emplear y porqué emplear cada uno.

Creo que es muy importante evitar al máximo el código acoplado, es decir, las capas tienen que estar claramente identificadas y un cambio en una de las capas debería afectar lo mínimo posible a las demás. Un ejemplo muy claro es el uso de un ORM que nos facilita que el uso de una u otra base de datos sea indiferente.

Este punto creo que es muy importante porque nos permite hacer mejoras de cualquier tipo de form más sencilla, por ejemplo cambiar la base de datos, un framework, etc.

He aprendido la importancia de elegir un Framework adecuado. Nos proporciona, además de una base para porder mejorar el tiempo de producción de una aplicación, una base testeada y preparada para un entorno de producción.

Otro de los aspectos más importantes aprendidos durante la elaboración del PFC ha sido en cuanto al tiempo. Una buena planificación lo es todo, te asegura unos plazos de tiempo más o menos fijos, proporciona una guía a lo largo de todo el proyecto que hace que no nos deviemos más de lo necesario.

Mi mayor problema en el proyecto ha girado en torno a la planificación, en primer lugar no he podido dedicar el tiempo deseado durante todo el proyecto, teniendo periodos de tiempo de inactividad. Otro de los problemas que he tenido es el haberme salido un poco de la planificación para ver alternativas, otros Frameworks, otros clientes, etc. Ahora, me parece un gran error. El análisis de la aplicación se realiza al principio y se debe seguir las reglas establecidas y salvo error o causa de fuerza no se debe perder tiempo en operaciones que retrasen el proyecto.

La desviación de tiempo es dificil de cuantificar debido a los periodos de inactividad, que he ido supliendo con periodos de gran actividad, diez horas diarias o más. Hablando en terminos directo de fecha de planificación y fecha de entrega he tenido varios meses de retraso.

9.3. Trabajo futuro

Puesto que el único objetivo de la aplicación era el apredizaje, ahora abro dos vías:

- **Software libre:** Subiré el repositorio a cualquier GIT público de forma que cualquier persona pueda contribuir. Añadiré una pequeña versión web con Angular JS y Bootstrap para que también se vaya avanzando en esa línea.
- **Publicación de la aplicación:** Subiré la aplicación a Google Play en cuanto disponga de una versión Web que apoye a la versión Android.

Bibliografía

[Apache(2015)] Apache. Tomcat 7. http://tomcat.apache.org/tomcat-7.0-doc/, 2015.

[Cascales and Lucas(2000)] Bernardo Cascales and Pascual Lucas. *LaTeX, una imprenta en sus manos*. A. D. I., 2000. ISBN 978-0-13-142246-9.

[CCOO(2010)] CCOO. Tablas salariales 2010 IV Convenio Colectivo. http://www.uca.es/sindicato/ccoo/documentos/tabla-salarial-pas-laboral-2010.pdf, 2010.

[Debian(2015)] Debian. Raspbian, Debian + Raspberry Pi. http://www.raspbian.org/RaspbianDocumentation, 2015.

[Deepak Alur and Malks(2008)] Jonh Crupi Deepak Alur and Dan Malks. *J2EE Patterns, Best Practices and Design Strategies*. Prentice Hall, 2008. ISBN 978-0-13-142246-9.

[Felker(2011)] Donn Felker. *Android Appplication Development for Dummies*. WILEY, 2011. ISBN 978-0-470-77018-4.

[Finegan and Liguori(2012)] Edward Finegan and Robert Liguori. *OCA Java SE 7*. Mc Graw Hill, 2012. ISBN 978-0-07-178942-4.

[Google(2015)] Google. Android SDK. https://developer.android.com/guide/index.html, 2015.

[Hibernate(2015)] Hibernate. Hibernate documentation. http://docs.jboss.org/hibernate/orm/4.3/manual/en-US/html/, 2015.

[Jenkins(2015)] Jenkins. Jenkins, CI. https://wiki.jenkins-ci.org/display/JENKINS/Use+Jenkins, 2015.

[JUnit(2014)] JUnit. Test Unitarios Java, JUnit. http://junit.org/project-info.html, 2014.

[Mockito(2014)] Mockito. Mocks para JUnit, Mockito. http://mockito.org/, 2014.

[Robotium(2014)] Robotium. Test para Android. http://robotium.com/pages/user-guide, 2014.

[SonarQube(2015)] SonarQube. SonarQube. http://docs.sonarqube.org/display/SONAR/Documentation, 2015.

[Spring(2015)] Spring. Spring documentation. http://spring.io/guides, 2015.